UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Módulo de compilación online

Fco. Javier Ibarra Ocaña
Tutor: Alejandro Sierra Urrecho

Febrero 2018

Módulo de compilación online

AUTOR: Fco. Javier Ibarra Ocaña
TUTOR: Alejandro Sierra Urrecho

Dpto. Ingeniería informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Febrero de 2018

Resumen (castellano)

El aprendizaje online está en auge. La principal característica de este tipo de cursos permite a los estudiantes adaptarse a la velocidad de aprendizaje que ellos mismos necesiten. Otra característica muy importante es que, al realizar el curso a través de un navegador, normalmente no se necesitan herramientas adicionales para cursarlo.

Este Trabajo de Fin de Grado consiste en la realización de un módulo de compilación y ejecución para la plataforma C-Learning. Esta plataforma permite realizar un curso interactivo de introducción al lenguaje de programación C.

Actualmente la aplicación se encuentra en funcionamiento, pero con ciertas limitaciones de las cuales algunas se tratarán en este trabajo mientras que otras se desarrollarán en un futuro.

La funcionalidad que corresponde a este trabajo de fin de grado incluye la compilación y ejecución de código C en el servidor donde se encuentre la aplicación. A esta funcionalidad se le añadirán mecanismos que permitan al usuario guardar su progreso y descargar el trabajo realizado. De esta forma se pretende eliminar todo tipo de tareas de configuración de un entorno de desarrollo por parte del usuario que realice el curso.

El objetivo que se persigue eliminando las tareas de configuración es la reducción de la curva de aprendizaje cuando se aprende el primer lenguaje de programación. De esta forma se permite que el alumno pueda centrarse exclusivamente en aprender el lenguaje de programación y en cómo utilizarlo.

También se han realizado mejoras en el diseño general de la aplicación y en varios módulos ya existentes para que sean compatibles entre sí.

Adicionalmente se han seguido una serie de estrategias de desarrollo que pretenden asegurar que en el futuro si se desean realizar cambios en la aplicación, estos puedan realizarse sin grandes problemas.

Palabras clave (castellano)

Laravel, WebSocket, Compilador, C, REST, Ace, GCC, Curso online

Abstract (English)

Online learning is growing up very fast. The main feature of these courses is to ease the way to adapt the content to the user skills. The online courses provide other important aspects, like avoiding users to need anything else than a browser.

The key objective of this Bachelor Thesis is to deliver a new module for C-Learning platform that allows users to compile and execute C code into the browser.

C-Learning is a web application to create and do online C courses.

The current web application is working with some restricted features. Some of these characteristics have been redesigned and the others will be in the future.

The main hallmark in the scope of this Bachelor Thesis are compiling and execute C code in the server side. This allows users to not have to configure a development environment. There are some secondary features implemented that deliver users a way to save their work and download it.

With these features, it is possible to avoid the configuration tasks by users and reduce the learning curve in the first development language learned. In this way, it allows students to focus exclusively on learning the language, and how to use it.

Additionally, the main design has been improved as some existing modules need to ensure full compatibility between them.

Finally, some development strategies have been followed to allow to insert future changes into the web application without problems.

Keywords (inglés)

Laravel, WebSocket, Compiler, C, REST, Ace, GCC, Online course

Agradecimientos

Tras pasar por un curso de formación profesional de informática mis padres, y mis amigos me animaron a realizar una carrera, acepté. Hoy puedo decir que no me arrepiento de esa decisión, y que probablemente sea una de las mejores que he tomado en mi vida. No ha sido fácil, sin embargo, he tenido muchos compañeros y algunos se han convertido en buenos amigos, con los que he compartido el día a día estos últimos años.

Oscar, Micho, Cristian, terminaron antes que yo, pero han dejado huella en mí. Adrián y César vinieron después pero también para quedarse. Por supuesto no podría faltar Celia, que ha estado ahí siempre, en los buenos y en los malos momentos, a ella le dedico este trabajo especialmente. No pueden faltar los que decidieron que la informática no era lo suyo, María, Inés y Carlangas, os fuisteis, pero solo de la carrera.

A mis padres, Curra y Javier, por animarme a hacer lo que me gusta y a conseguir todo lo que me proponga.

A mis amigos Kike, Carlos, Raúl y por supuesto no me puedo olvidar de Nistal.

Por último, agradecer a Alejandro que me diese la oportunidad de realizar este trabajo.

A todos vosotros, Gracias.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

En la actualidad la utilización de las tecnologías de la información se está convirtiendo en un elemento indispensable en el sector educativo. Crear y mejorar herramientas que permitan mejorar la progresión del aprendizaje es fundamental.

El principal problema que se encuentran los estudiantes que cursan algún tipo de ingeniería en el que se requiera aprender a programar, es la abrumadora cantidad de tareas que hay que realizar antes de empezar a programar. Es necesario configurar un entorno de programación según el lenguaje que se desee aprender, lo cual, en muchas ocasiones no es algo trivial.

Hay que tener en cuenta el sistema operativo en el que se desea desarrollar y para el que se desea desarrollar, no es lo mismo programar una distribución de Linux que Windows. A parte del sistema operativo, es necesario un editor o un IDE (Integrated Development Enviroment) para escribir el código. Y por último será necesario un compilador o un intérprete (según el lenguaje) para poder crear los ejecutables si es necesario y probar nuestro código.

Para poder evitar estos problemas, la idea de tener un entorno disponible, independiente del sistema operativo y que lo único que requiera es tener un navegador instalado para empezar a aprender, es muy interesante.

La principal motivación de este proyecto es la creación de un módulo en una aplicación de aprendizaje online que permita compilar y ejecutar código C, de forma que el alumno que vaya a usar esta aplicación no tenga que involucrarse en la configuración del entorno de desarrollo.

La decisión de que la aplicación trabaje con código C es que es el primer lenguaje que se enseña en la Universidad Autónoma de Madrid. Además, a pesar de las limitaciones que tenían las primeras versiones de la aplicación, fue empleada con éxito durante el curso académico 2016-2017 en la asignatura de Programación 1 del Grado de Ingeniería de Tecnologías y Servicios de Telecomunicación.

1.2 Objetivos

El principal objetivo de este trabajo de fin de grado es la realización de un módulo de compilación y ejecución online para una aplicación de aprendizaje en C. Esto incluye realizar un diseño apropiado a partir de la aplicación actual y llevar a cabo el desarrollo del mismo.

El objetivo de este módulo es permitir compilar y ejecutar cualquier código escrito en C utilizando las librerías estándar y ejecutarlo de forma dinámica, esto es que permita interacción con el usuario.

Para la realización con éxito de este proyecto será necesario cumplir los hitos descritos a continuación:

- Análisis de la aplicación actual. Dado que este proyecto no parte de cero, se deberá analizar cuidadosamente la aplicación existente, así como las tecnologías utilizadas.
- Realización de un diseño acorde a la aplicación estudiada. Este diseño se centrará en modificar el módulo de complicación y ejecución antiguo y realizar cambios en algún otro módulo si fuese necesario intentando que estos sean mínimos.
- Una vez realizado el diseño se desarrollarán las modificaciones necesarias para que el nuevo módulo se pueda desarrollar.
- El siguiente paso es desarrollar el módulo en sí.
- Posteriormente se realizarán pruebas en el entorno de desarrollo.

1.3 Organización de la memoria

Esta memoria está dividida de la siguiente forma:

- En primer lugar, se analizará el estado del arte y las tecnologías que se utilizarán en el desarrollo del trabajo de fin de grado.
- En segundo lugar, se realizará un diseño en base a la situación actual de la aplicación y a las tecnologías definidas en el apartado anterior.
- En tercer lugar, se describirá el desarrollo del trabajo realizado.
- En cuarto lugar, se definirán las pruebas realizadas y los resultados obtenidos.
- Por último, se analizará el trabajo realizado y se sacarán las conclusiones a partir del mismo, así como se podría mejorar en un futuro la aplicación desarrollada y como se le podría dar un uso adecuado.

2 Estado del arte

Para realizar un diseño del módulo de compilación se va a realizar un análisis de distintas plataformas que permiten realizar una funcionalidad similar a la que se pretende desarrollar.

2.1 Análisis sobre plataformas online de compilación en C

C es un lenguaje de programación imperativo y estructurado de propósito general diseñado por Dennis Ritchie en 1972. El primer estándar ANSI C no fue ratificado hasta 1989 y publicado en 1990. A día de hoy se han publicado cuatro versiones del estándar, siendo la última de 2011 [1].

Para poder ejecutar un programa escrito en C, es necesario generar un ejecutable. Este ejecutable se consigue en primer lugar compilando el código C. Esto generará un objeto no ejecutable. En segundo lugar, hay que enlazar el objeto anterior para generar un fichero binario ejecutable [2]. Para realizar estas tareas se necesita un programa que compile y enlace como puede ser *GNU Complier Collection* (GCC).

Una plataforma online de compilación permite generar un ejecutable a partir de un código, ya sea dado en un fichero o escrito directamente en un navegador. Varios ejemplos que podemos encontrar en Internet son, TutorialsPoint o CodeChef. Ambos no solo permiten compilar y enlazar código C, sino que también dan la posibilidad de ejecutar el resultado. Esto permite liberar completamente al usuario de tener que configurar un entorno de desarrollo en su propio equipo.

Para configurar un entorno de programación orientado al desarrollo de programas escritos en C existen múltiples alternativas. En primer lugar, es necesario saber sobre qué sistema operativo se va a desarrollar. Si se elige cualquier distribución de Linux es posible instalar de forma nativa el compilador GCC por ejemplo y ejecutar desde la línea de comandos. En cuanto al código se podría desarrollar en un editor de texto o en un entorno como NetBeans o Eclipse. Por otro lado, si se escoge Windows como entorno de desarrollo, será necesario usar u otro compilador que se pueda instalar o un método alternativo para instalar GCC. Es posible hacer esto mediante herramientas que ofrezcan funcionalidades similares a un entorno Linux como puede ser Cygwin [3]. Esto añade aún más complejidad a la configuración del entorno de desarrollo y eso es precisamente lo que se quiere evitar.

2.2 Análisis sobre las tecnologías utilizadas

2.2.1 REST

Representational state transfer (REST) es una arquitectura para diseñar y desarrollar servicios web. Está basada en la representación de objetos (HTML, CSS, PNG, etc) mediante acciones (GET, POST, PUT, DELETE) y URIs [4]. Una URI es una ruta que indica donde se encuentra el recurso.

La utilización de REST nos permitirá realizar un desarrollo sencillo del nuevo módulo, fácilmente escalable y robusto ya que se podrán realizar pruebas independientes tal y como se verá en el apartado 5.

2.2.2 Laravel

Laravel es un framework PHP que adopta la arquitectura REST para el desarrollo de aplicaciones web. Laravel incluye otras tecnologías que complementan el framework. Entre ellas cabe destacar Bootstrap. Bootstrap es otro framework basado en HTML, CSS y JavaScrirpt [5] orientado al desarrollo flexible y adaptativo de páginas web.
Por otro lado, Laravel facilita realizar un diseño basado en el patrón de diseño Modelo Vista Controlador (MVC). Concretamente, se pueden realizar vistas que mostrarán la información que podrán asociarse con uno o varios controladores que permitirán añadir la funcionalidad necesaria para cumplir los requisitos deseados.

Figura 2-1: Logo de Laravel

Otra característica de Laravel es que contiene un constructor de consultas. Este constructor permite trabajar con la base de datos de la aplicación de forma sencilla y transparente. Si en algún momento es necesario cambiar la base de datos (no el diseño sino el tipo) no será necesario cambiar nada en el código, solo en los ficheros de configuración.

A parte del framework en sí, los desarrolladores de Laravel proporcionan un entorno de desarrollo preconfigurado que contiene todas las herramientas básicas para empezar a desarrollar un proyecto utilizando el framework [6].

2.2.3 Websocketd

Websocketd es una herramienta que permite comunicar una aplicación a través de la línea de comandos utilizando el protocolo WebSocket. Esta herramienta necesita utilizar un ejecutable y enlazarlo con un socket en un determinado puerto. A partir de este punto, todo lo que venga a través del socket se enviará a STDIN y si el programa necesita escribir algo lo hará en STDOUT del cual el socket tendrá que leer para enviar su contenido [7].

El protocolo WebSocket permite establecer una comunicación de doble sentido entre un cliente y un servidor. El objetivo principal de este protocolo es que esa comunicación se realice a través de aplicaciones web [8].

2.2.4 Ace

Ace es un editor de código escrito en JavaScript [9]. Permite editar código de forma sencilla ya que resalta la sintaxis según el lenguaje para el que se configure y proporciona la generación de sangrías automáticas.

3 Diseño

En este apartado se explicará el modelo de vida del software empleado, así como los requisitos necesarios para el nuevo módulo. Se realizará un análisis para comprobar cuál es el estado actual de la aplicación para, en función de ese estado realizar un diseño adecuado.

3.1 Modelo de ciclo de vida del software

El módulo a implementar en conjunto con la aplicación en general sigue el patrón de diseño modelo vista controlador (MVC a partir de ahora). Utilizando esto como base se ha elegido un ciclo de vida de tipo incremental para el desarrollo del software.

El desarrollo incremental permite dividir un proyecto en pequeñas partes que se irán desarrollando de forma independiente. Estas partes (o fases) se irán ampliando hasta conseguir la funcionalidad deseada. Una vez realizada una fase se probará de forma individual (pruebas unitarias) y con la aplicación (pruebas de integración).

3.2 Análisis de requisitos

En este apartado se citan los requisitos tanto funcionales como no funcionales que deberán cumplirse para que el módulo de ejecución sea completamente operativo.

Requisitos funcionales:

RF1: El usuario podrá crear nuevos ficheros. Las extensiones de los ficheros podrán ser:
   a. .c, ficheros que se podrán compilar.
   b. .h, ficheros de cabecera.
   c. .txt, ficheros de entrada y/o salida de datos.

RF2: El usuario podrá borrar los ficheros que cree. No se podrán borrar ni el fichero main.c ni los ficheros adicionales que se definan para el objetivo que se esté realizando.

RF3: Guardar datos. Por defecto, el contenido de los editores que permiten la modificación de los ficheros no se guarda automáticamente por lo que es necesario guardar los datos (en el servidor) de forma manual.

RF4: Descargar ficheros. El usuario podrá descargar los ficheros de cada objetivo individualmente. Estos ficheros deberán comprimirse en un fichero .zip.

RF5: Compilación. El usuario podrá compilar cada objetivo de forma individual mediante un botón. El resultado de la compilación se mostrará en una salida de texto que simula un terminal.

RF6: Ejecución. El usuario podrá ejecutar el código mediante un botón. El resultado de la ejecución se mostrará en una salida de texto que simula un terminal. Si la ejecución es interactiva, es decir, requiere que el usuario introduzca valores, éstos podrán introducirse desde un campo de texto.

RF7: El usuario podrá “limpiar” la salida de texto.

Requisitos no funcionales:

RNF1: Se garantizará que el usuario no pueda ejecutar un programa durante una ejecución existente. De esta forma se bloquearán los botones de ejecución y de compilación mientras haya una ejecución en curso.

RNF2: De igual forma que en el requisito no funcional anterior (RNF1), no se podrá compilar ni ejecutar mientras una compilación esté en curso.

RNF3: Se garantizará que no se ejecute ni se compile la función system() por motivos de seguridad.

RNF4: Se garantizará que al abrir un fichero mediante fopen() la ruta en la que se aloje dicho fichero estará en una carpeta dedicada al propio usuario que lo ejecute.

RNF5: Se mantendrá el uso de la arquitectura REST siempre que sea posible.

3.3 Análisis de los módulos a modificar

Existen dos módulos que requieren modificaciones. El módulo de compilación y ejecución, que se volverá a desarrollar por completo atendiendo a los requisitos mencionados en el punto 3.2. El módulo de edición deberá modificarse para atender a las necesidades del módulo de compilación y ejecución. Estas necesidades se describen a continuación.

3.3.1 Modificaciones en el módulo de edición

Modificaciones en la base de datos:
El actual diseño de la base de datos permite guardar un único fichero de código para cada objetivo. Esto no es práctico si se necesitan varios ficheros para definir un objetivo. Por ese motivo es necesario realizar una nueva tabla que permita guardar el código de cada objetivo tanto para objetivos con un único fichero como para varios ficheros.

A partir de este punto se han mejorado las tablas existentes para que el almacenamiento de la información sea más consistente y ésta se pueda manejar de forma dinámica.

El diseño de partida sería el que se muestra en la Figura 3-1:

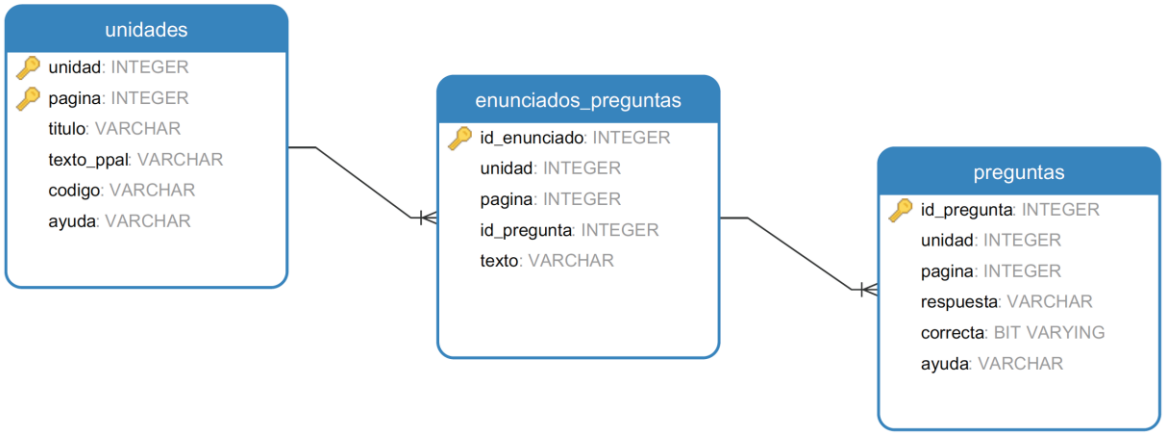


Figura 3-1: Diagrama entidad relación antiguo

Las tablas que se van a modificar son: unidades, enunciados_preguntas y preguntas.
En primer lugar, el nombre de las tablas es poco descriptivo ya que por ejemplo la tabla preguntas no alberga preguntas sino respuestas, enunciados_preguntas contiene la pregunta en sí y unidades contiene los enunciados de los objetivos. Así mismo se han reorganizado las columnas de dichas tablas para que la información se encuentre en el lugar adecuado.

El nuevo diseño quedaría como se muestra en la Figura 3-2:

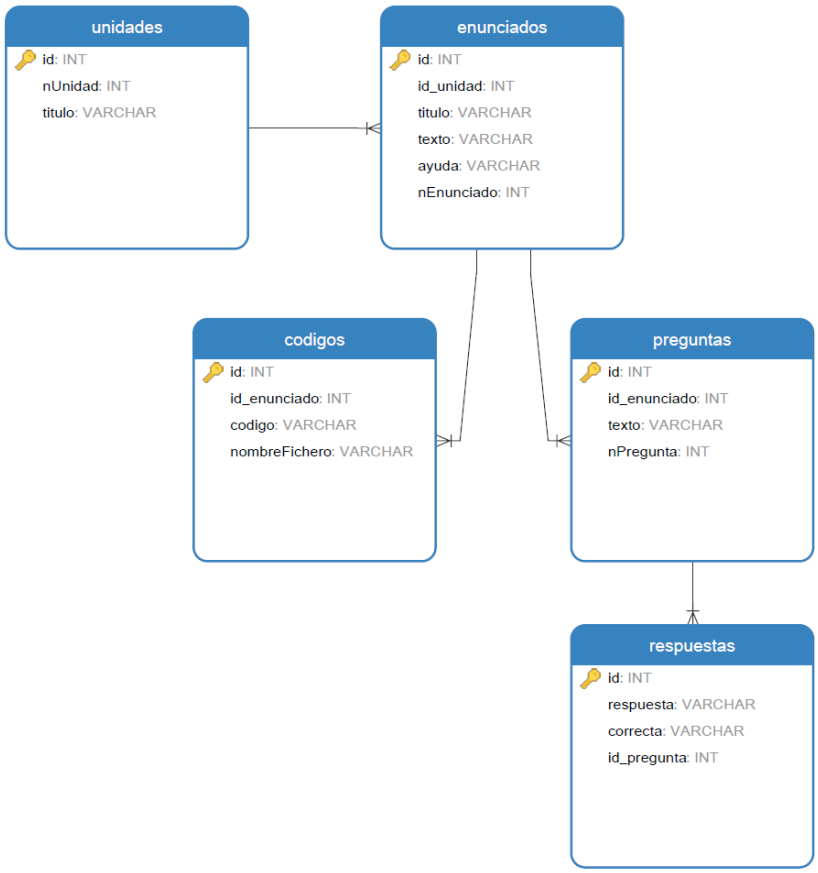


Figura 3-2: Diagrama entidad relación nuevo

- *Unidades:* La tabla anterior se ha modificado permitiendo solo almacenar el número de la unidad y el título. Hay que añadir que el número de la unidad es distinto del identificador de la unidad. Aunque ambos son únicos el primero es modificable por parte del docente (o administrador) mientras que el segundo sirve únicamente para crear relaciones con otras tablas.

- *Enunciados:* La tabla enunciados es nueva y tiene como objetivo almacenar cada uno de los objetivos de cada unidad. Esta tabla guardará un texto para el enunciado del objetivo; un texto de ayuda; un título para el enunciado y un número de enunciado. Al igual que en la tabla unidades, existe un identificador que es distinto que el número de ejercicio.

- *Preguntas:* Esta tabla sería equivalente a la anterior tabla enunciados_preguntas salvo que no necesita guardar un identificador de la unidad a la que pertenece.

- *Respuestas:* Esta tabla es equivalente a la tabla preguntas salvo que ya no se guarda ningún texto de ayuda (el texto de ayuda es global para cada objetivo y se guarda en la tabla enunciados).

- *Códigos:* Esta tabla es nueva y guardará el código perteneciente a un fichero. También guardará el nombre de dicho fichero y el identificador correspondiente a un objetivo (al que pertenecerá).

Modificaciones en la interfaz

Para poder guardar los contenidos en la base datos conforme al nuevo diseño habrá que modificar la interfaz que permite la introducción de la información. El principal cambio en este punto es que la interfaz para crear unidades y objetivos se divide en dos formularios separados. Dentro de cada uno se mantendrá el diseño en cuanto a la distribución de los campos de entrada. Será necesario añadir un campo de texto para introducir un nombre a la hora de crear un nuevo fichero y un botón para crearlo.

Por último, se realizarán cambios en el estilo de la página para que sea más intuitiva y permita el acceso desde diferentes tipos de pantalla.

Modificaciones en el diagrama de clases

Para utilizar la información que se guarde en la base de datos deberán modificarse las clases existentes para que se adapte al diseño de la nueva base de datos. El nuevo diseño de clases se muestra en la Figura 3-3.

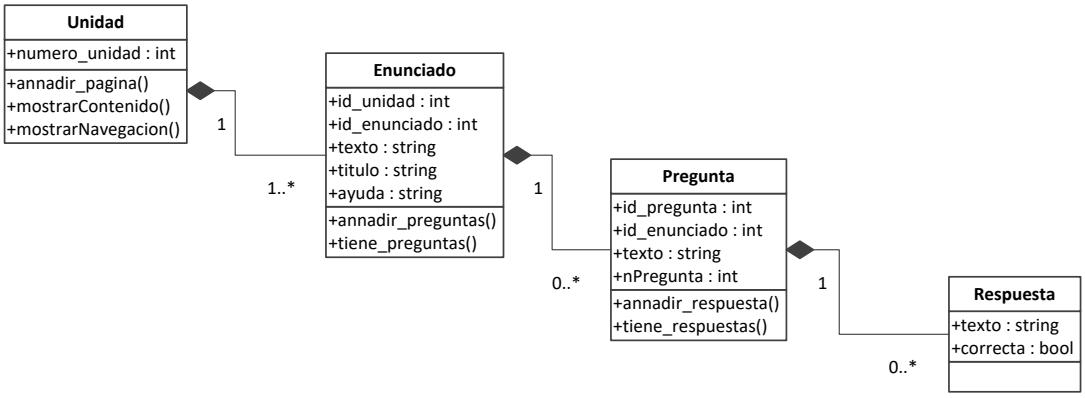


Figura 3-3: Diagrama de clases

3.3.2 Modificaciones en el módulo de compilación y ejecución

Una vez modificada la base de datos para el módulo de edición no se requerirán más modificaciones para el módulo de compilación y ejecución en la misma.

Modificaciones en la interfaz

Para poder implementar los requisitos funcionales definidos en la sección 3.2 deberán realizarse las siguientes modificaciones. Se añadirán nuevos botones para poder realizar dichos requisitos, guardar, descargar y crear un nuevo fichero. Todos ellos junto a los ya existentes que permiten compilar y ejecutar. También se añadirá un campo de texto para dar un nombre a los ficheros que se creen de igual forma que en el editor de contenidos.

Modificaciones en los controladores

Siguiendo el MVC como patrón de diseño en conjunto con la arquitectura REST en la que está basada la aplicación, la funcionalidad descrita en los requisitos (sección 3.2) deberá implementarse en el controlador de la vista. En la primera versión dicha funcionalidad se encuentra implementada en un fichero aparte que no cumple estas condiciones.

Para la comunicación síncrona que se produce durante una ejecución del programa que pueda implementar el usuario, la arquitectura REST no es adecuada ya que no permite este tipo de comunicación. Por ese motivo se ha decidido utilizar WebSockets para implementar este requisito. Mediante el uso de WebSockets es posible establecer una comunicación interactiva entre el navegador del cliente y el servidor [10].

Modificaciones en el árbol de directorios

Para poder asegurar que los alumnos puedan guardar sus soluciones a los objetivos propuestos en cada unidad se ha modificado la estructura de directorios conforme a esta necesidad. En la versión anterior se creaba una carpeta por alumno que contenía los ficheros que se compilaban, así como los ejecutables, pero en ningún momento se podía asegurar a que objetivo pertenecían los ficheros que se ejecutaban o compilaban. Teniendo esto en cuenta se ha añadido al sistema de directorios una nueva estructura. Cada usuario dispondrá de una carpeta personal, que esta a su vez contendrá una carpeta por cada unidad y cada carpeta de cada unidad contendrá una por cada objetivo que tenga esa unidad. De esta forma los ficheros de cada objetivo se guardarán en su carpeta correspondiente. Esto

permitirá al usuario volver a un objetivo anterior a consultar la solución que propuso o a descargar sus ficheros.

Figura 3-4: Nuevo árbol de directorios

4 Desarrollo

En la fase de desarrollo del proyecto se realizó la implementación del módulo de compilación y ejecución y se llevaron a cabo las modificaciones indicadas en la sección de diseño en el módulo de edición de unidades.

4.1 Modificaciones en el módulo de edición de unidades

Las modificaciones en este módulo se pueden diferenciar en tres apartados: modificación de la base de datos, modificación de las clases que gestionan la información y los controladores y por último la modificación de la interfaz.

4.1.1 Modificación de la base de datos

La modificación de la base de datos se ha realizado en base al diseño propuesto en la sección 3.3.1. Para ello, el framework Laravel dispone de un sistema de migraciones que permite la modificación de una base de datos durante el desarrollo de un proyecto [11]. Este sistema de migraciones actúa como un control de versiones para la base de datos por lo que permite realizar un rollback en caso de que algo vaya mal. En este sentido Laravel aporta una forma muy sencilla de modificar la base de datos sin utilizar sentencias SQL directamente.

```
13      public function up()
14      {
15          Schema::create('unidades', function (Blueprint $table) {
16              $table->increments('id')->unsigned();
17              $table->integer('nUnidad');
18              $table->string('titulo');
19              $table->unique(['id', 'nUnidad']);
20          });
21      }
```

Figura 4-1: Ejemplo de migración. En este caso de la tabla unidades.

4.1.2 Modificación de la lógica del módulo de edición

La modificación en la lógica del módulo de edición se puede separar en dos puntos: la modificación de las clases y la modificación de los controladores.

Modificación de las clases:
Las clases dentro del proyecto en general se utilizan para manejar los datos de la aplicación. El desarrollo de las nuevas clases se basa en el diseño de la base de datos por lo que permiten guardar los datos que procedan de la base de datos, así como manipularlos cuando sea necesario.

Modificación de los controladores:
El mayor cambio en la lógica del módulo de edición se encuentra en la parte de los controladores. Esto se debe básicamente a que los controladores son quienes se encargan de manejar la información entre la interfaz y la base de datos. En los controladores se han modificado los accesos a la base de datos, esto incluye la forma de insertar los datos (*inserts*), la forma de actualizar los datos (*updates* y *deletes*) y la forma de obtener la información necesaria para cada situación (*selects*). Para realizar los accesos a la base de datos se ha utilizado el constructor de consultas que proporciona Laravel. Este constructor permite crear consultas SQL de una forma sencilla sin tener que usar SQL “puro” dentro

del código. Como nota adicional sobre este constructor de consultas, añade un *PDO parameter binding* [12] que protege la aplicación de ataques de inyección SQL.

```
51
52          $query = DB::table('progress')
53                          ->select('unit','target')
54                          ->where('user_id', '=', $id)
55                          ->get();
56
```

Figura 4-2: Select utilizando el Query Builder de Laravel

4.1.3 Modificación de la interfaz

La modificación de la interfaz del módulo de edición parte de la división de los formularios en mencionados en el apartado 3.3.1. De esta forma queda bien definida la forma de crear una unidad y un objetivo. Se ha mantenido la distribución de los elementos de la misma forma que en la versión anterior. Sin embargo, se ha cambiado el estilo para que se adapte mejor a diferentes tipos de pantallas. También se han añadido etiquetas y *tooltips* para dar más información al usuario acerca del uso de cada parte del formulario.

Como característica adicional se ha añadido una lista en forma de acordeón para ocultar las preguntas, de forma que si no se está trabajando sobre la inclusión de una pregunta se puede ocultar para obtener una mejor visibilidad del resto de elementos.

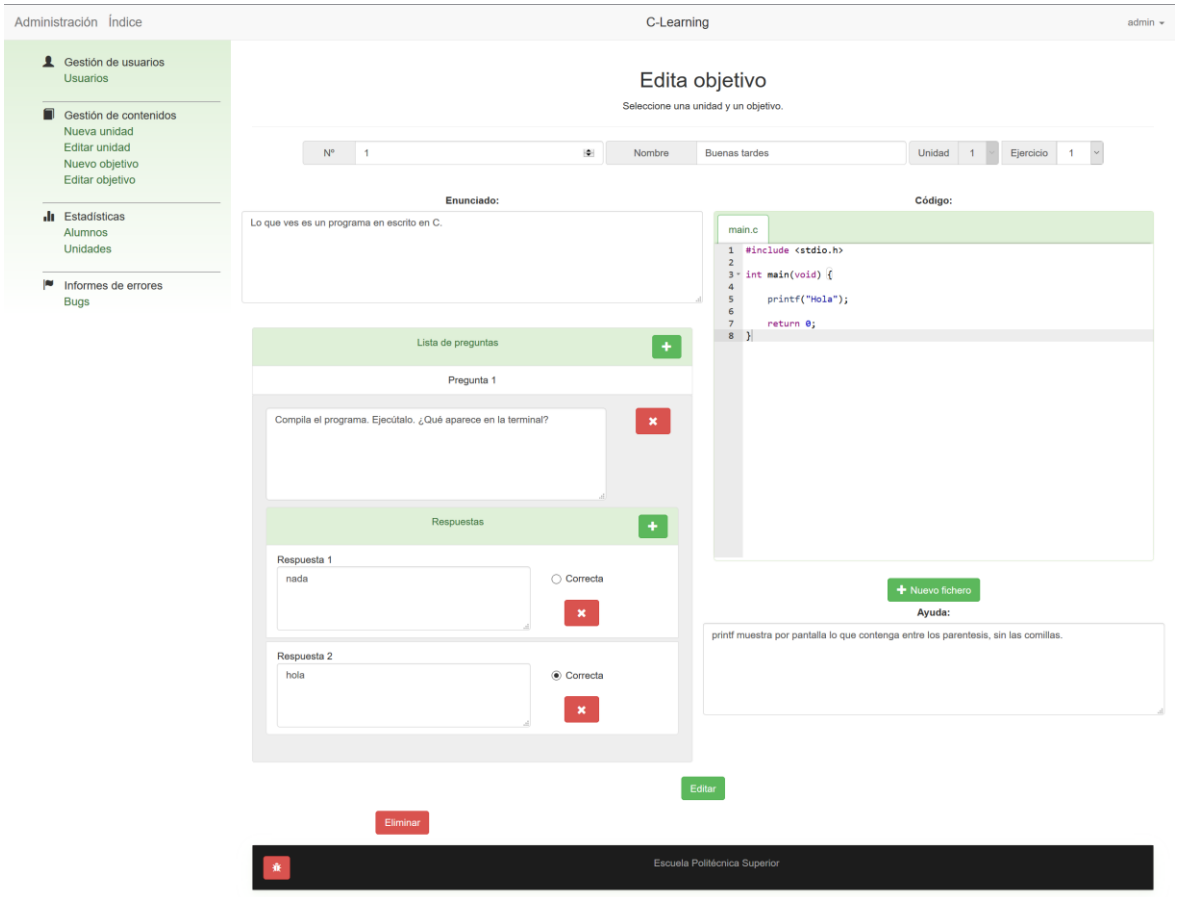


Figura 4-3: Nueva interfaz del editor de ejercicios.

4.2 Modificaciones en el módulo de compilación y ejecución

El módulo de compilación y ejecución de la versión anterior estaba prácticamente obsoleto por lo que se ha modificado completamente en cuanto a la lógica se refiere. En cuanto a la interfaz se han añadido nuevos elementos para implementar toda la funcionalidad descrita en el apartado 3.2. Se ha modificado ligeramente la parte visual para que sea un poco más intuitivo el manejo de la misma por parte de los usuarios y se adapte mejor a diferentes tamaños de pantalla.

4.2.1 Modificación de la lógica del módulo de compilación y ejecución

Este módulo implementa la siguiente funcionalidad:

Compilación
Para compilar un programa de un objetivo el usuario deberá utilizar el botón *Compilar*. Tras hacer esto se llamará al recurso:

/unidad{id}/ejercicio{ejercicio}/user{usuario}/compilar

Es necesario, a la hora de llamar al recurso, saber qué usuario lo va a utilizar y en qué unidad y objetivo se encuentra. Con esta información se definirá el directorio actual de trabajo. Una vez se sepa se podrán recopilar los ficheros que se están utilizando y se pasará a compilar. La compilación se realiza en tres pasos:

El primero es compilar el código perteneciente al usuario. Si no existen errores se pasará a la siguiente fase. Si existen errores estos se notificarán al usuario y se omitirán los siguientes pasos.

En segundo lugar, se realizará un análisis del código. El objetivo es encontrar sentencias que incluyan las funciones *fopen()* y *system()* para mejorar la seguridad.

- **fopen():** El problema principal que acarrea *fopen* es que al intentar abrir un fichero se buscará en el directorio en el que se ejecute. Teniendo en cuenta la arquitectura del framework, esto se hace desde la carpeta *public*, sin embargo, las carpetas de los usuarios se encuentran en *storage/userprograms*.
  Para resolver este problema se buscarán sentencias de código que incluyan *fopen* y se formateará la ruta añadiendo la ruta en la que está trabajando el usuario, utilizando como referencia el número de la unidad y el número del objetivo.

- **system():** *system* permite ejecutar un comando externo, por ejemplo *ps*. Esto supone un grave problema de seguridad ya que cualquiera podría ejecutar cualquier sentencia que pueda dañar al servidor. Se podría por ejemplo borrar la base de datos, alterarla o borrar ficheros de otros usuarios, pero también se podría dañar el sistema. Por estos motivos de la misma forma que se buscan sentencias con *fopen* se buscan llamadas a la función *system*. Si se encuentra un *system* se interrumpirá la compilación y se mostrará un mensaje al usuario para que no la utilice.

En ambos caso se han tenido en cuenta que, tanto system como fopen, sean llamadas a función y no se encuentren en una variable que tenga como objetivo ser solo una cadena de texto, como por ejemplo*, printf("system() no se puede utilizar");*

Un último lugar y tras realizar el análisis se generarán ficheros nuevos denominados *nombreFicheroOriginal.temp.c*. Estos ficheros se compilarán con el único fin de crear un ejecutable, que será el que utilice el usuario cuando desee ejecutar su programa. Los ficheros con extensión *.temp.c* se borrarán tras la compilación.

Ejecución

Una de las funcionalidades más importantes es la de ejecutar. Cuando el usuario pulse el botón ejecutar ocurrirá lo siguiente en un estricto orden:

1. El navegador enviará una petición al servidor que incluirá el id del usuario, el número de unidad, el número de ejercicio actual y el puerto en el que se establecerá la conexión interactiva.
2. El servidor comprobará que existe el ejecutable, de forma que la compilación se haya llevado a cabo correctamente. En caso contrario se notificará al usuario y se finalizará el proceso de ejecución.
3. Si existe el ejecutable, se abrirá un socket.
4. Como la tecnología de webscokets utilizada (*Websocketd*) asigna la recepción del socket al *stdin* y el envío al *stdout* es necesario deshabilitar el almacenamiento en el buffer de salida. Esto se hace utilizando el comando *stdbuf –o -0* (*-o* indica que se usa el buffer *stdout*, y *-0* lo deshabilita).
5. Por último, se ejecutará el programa del usuario en el socket.

Dado que cada socket necesita un puerto para operar, para que no existan conflictos entre las distintas conexiones cada puerto se elegirá conforme al identificador único de cada usuario a partir del puerto 9000. Por ejemplo, el usuario con id igual a 3 se le asignará el puerto 9003.

Dado que es posible que no se pueda establecer conexión con el socket debido a algún tipo de incidencia en la red, la aplicación informará al usuario si ocurre algún problema de este tipo mostrando un diálogo.

Una vez establecida la conexión el programa desarrollado por el usuario se ejecutará y se comunicará con el navegador a través del socket. La salida del programa se enviará al "terminal" del que disponemos en la pantalla del objetivo que estamos realizando. Si es necesario que el usuario interactúe con el programa podrá hacerlo a través del mismo "terminal".

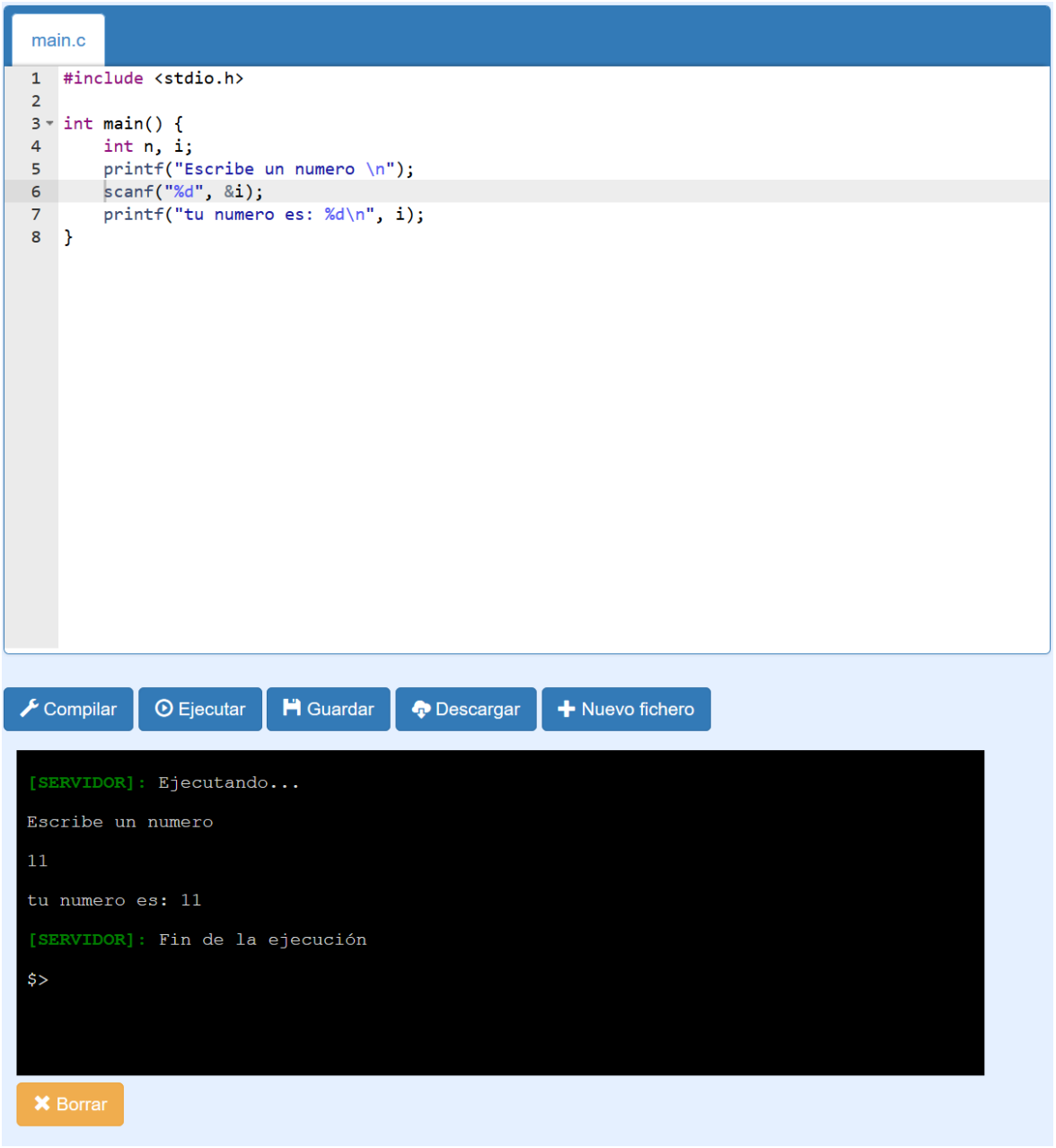


Figura 4-4: Ejemplo de ejecución interactiva.

Descarga de ficheros

El usuario puede descargar los ficheros de un objetivo utilizando el botón *Descargar*. Cuando se realice esta acción se enviará una petición de descarga al servidor. En este punto el servidor comprimirá los ficheros que existan en el directorio de trabajo actual (definido por la unidad y el objetivo en que se encuentre) y descargará el comprimido. La extensión del fichero comprimido será .zip y a la hora de comprimir se tiene en cuenta que no se comprima cualquier .zip existente en el directorio.

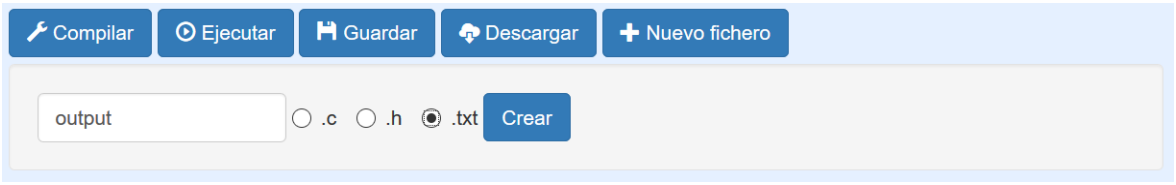
Guardar objetivo

La aplicación no guarda los ficheros del usuario automáticamente, por lo que se ha añadido la posibilidad de que el usuario guarde su trabajo utilizando el botón *Guardar*. Tras pulsarlo se notificará al usuario que se ha guardado su trabajo.

Otras funcionalidades

El usuario podrá crear ficheros utilizando la interfaz. Para ello deberá dar un nombre y elegir una extensión que puede ser .c, .h o .txt. Los nombres de los ficheros (con la misma extensión) no se pueden repetir. En el caso en que se intente crear un fichero que ya exista, la aplicación mostrará un error.

El usuario también podrá borrar ficheros. Para ello solo tiene que utilizar la *X* que se encuentra en cada pestaña. Si se pulsa, borrará el fichero correspondiente a esa pestaña.



El usuario también podrá borrar el contenido del "terminal" utilizando el botón *Borrar*.

5 Pruebas y resultados

Tras el desarrollo de los módulos es necesario someterlo un control de calidad y que permita conocer si la aplicación se comporta de forma adecuada ante situaciones inesperadas. Para ello se han realizado diversas pruebas que permiten comprobar el correcto funcionamiento de los módulos desarrollados.

Las pruebas que se han llevado a cabo son de caja negra. Este tipo de pruebas permiten probar el programa sin necesidad de interactuar con el código. El procedimiento para llevar a cabo este tipo de pruebas consiste en:

- Definir una entrada para la aplicación en un momento en concreto.
- Esperar una salida acorde con la entrada introducida o un error que indique al usuario que algo ha ido mal.

En algunos casos ya que las pruebas modifican la base de datos, se ha comprobado que la base de datos se ha modificado correctamente y no se han generado inconsistencias en los datos.

5.1 Pruebas en el módulo de edición

Se empezarán las pruebas por la edición de unidades. El punto de partida de la tabla de las unidades se muestra en la Figura 5-1

```
mysql> select * from unidades;
+----+-----------------------------------+---------+
| id | titulo                            | nUnidad |
+----+-----------------------------------+---------+
|  1 | Buenas tardes                     |       1 |
|  2 | Tipos de datos                    |       2 |
|  3 | Condicionales e iteraciones       |       3 |
| 10 | Arrays                            |       4 |
| 12 | El main no lo es todo: Funciones  |       6 |
| 16 | Ficheros de cabezera (.h)         |       8 |
+----+-----------------------------------+---------+
6 rows in set (0.00 sec)
```

Figura 5-1: Estado inicial de la tabla unidades

Crear una unidad

Para crear una unidad es necesario introducir todos los datos que están en el formulario de creación. El número de la unidad no puede estar repetido por lo que si se intenta crear se mostrará un error.

Se crea una unidad como se muestra en la Figura 5-2

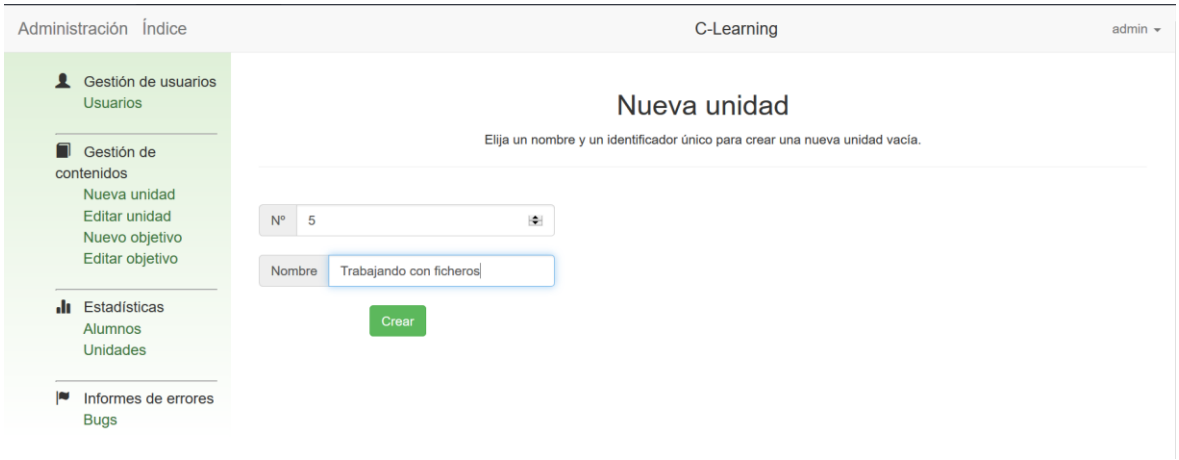


Figura 5-2: Creación de una nueva unidad

Y se comprueba que se ha creado la unidad

```
mysql> select * from unidades;
+----+----------------------------------+---------+
| id | titulo                           | nUnidad |
+----+----------------------------------+---------+
|  1 | Buenas tardes                    |       1 |
|  2 | Tipos de datos                   |       2 |
|  3 | Condicionales e iteraciones      |       3 |
| 10 | Arrays                           |       4 |
| 12 | El main no lo es todo: Funciones |       6 |
| 16 | Ficheros de cabezera (.h)        |       8 |
| 21 | Trabajando con ficheros          |       5 |
+----+----------------------------------+---------+
```

Figura 5-3: Resultado de crear una unidad correctamente

Como se puede ver en la última fila de la tabla, se ha creado la unidad de título Trabajando con ficheros y de número de unidad 5, tal y como se especifica en el formulario.

Editar una unidad

El procedimiento para modificar una unidad es similar al de crearla. La única diferencia es que hay que seleccionar la unidad que se desee modificar en la lista desplegable. Se probará a modificar la unidad anteriormente creada:

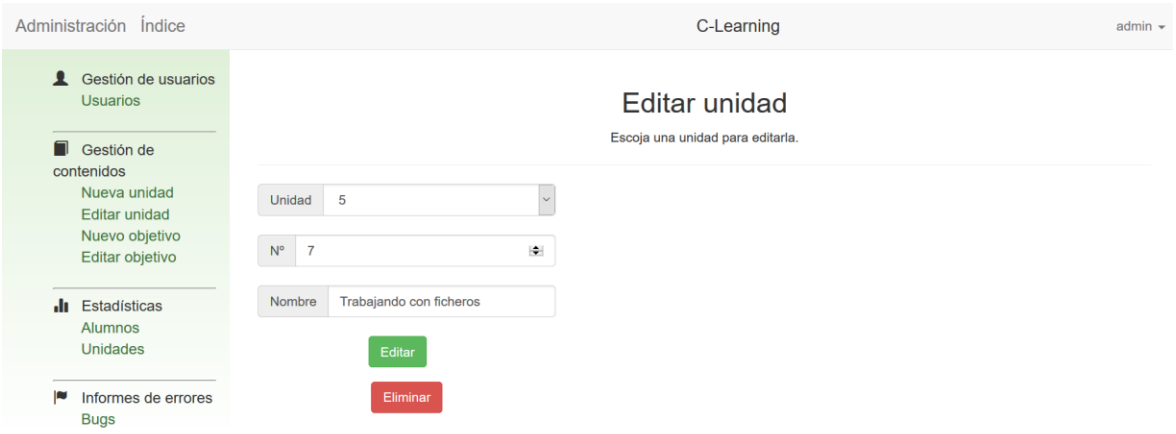


Figura 5-4: Edición de una unidad

En este caso hay que seleccionar la unidad 5. En los otros campos se cargarán los datos de la unidad, el número y el nombre. Tras cambiar el número de 5 a 7 tal y como se muestra en la Figura 5-4 y pulsar el botón Editar se puede comprobar en la base de datos el resultado que efectivamente se ha modificado el número de unidad (Figura 5-5):

```
mysql> select * from unidades;
+----+----------------------------------+---------+
| id | titulo                           | nUnidad |
+----+----------------------------------+---------+
|  1 | Buenas tardes                    |       1 |
|  2 | Tipos de datos                   |       2 |
|  3 | Condicionales e iteraciones      |       3 |
| 10 | Arrays                           |       4 |
| 12 | El main no lo es todo: Funciones |       6 |
| 16 | Ficheros de cabezera (.h)        |       8 |
| 21 | Trabajando con ficheros          |       7 |
+----+----------------------------------+---------+
7 rows in set (0.00 sec)
```

Figura 5-5: Resultado en la base de datos tras editar una unidad

Eliminar una unidad

Para eliminar una unidad basta con seleccionar una de la lista de igual forma que a la hora de modificar. Se va a borrar la unidad que se ha creado y modificado en las pruebas anteriores:

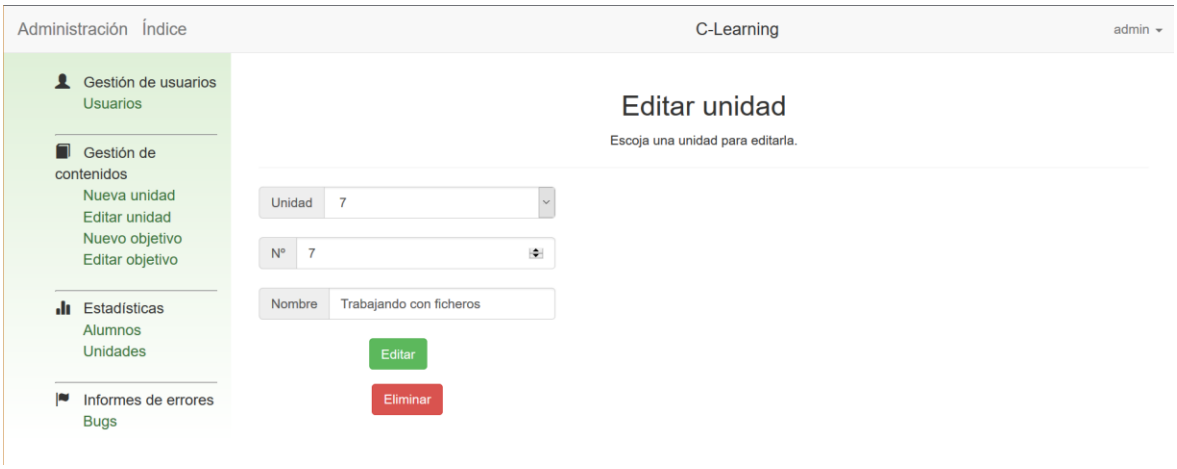


Figura 5-6: Selección para borrar una unidad

Se comprueba en la base de datos que efectivamente se ha borrado:

```
mysql> select * from unidades;
+----+-----------------------------------+---------+
| id | titulo                            | nUnidad |
+----+-----------------------------------+---------+
|  1 | Buenas tardes                     |       1 |
|  2 | Tipos de datos                    |       2 |
|  3 | Condicionales e iteraciones       |       3 |
| 10 | Arrays                            |       4 |
| 12 | El main no lo es todo: Funciones  |       6 |
| 16 | Ficheros de cabezera (.h)         |       8 |
+----+-----------------------------------+---------+
6 rows in set (0.00 sec)
```

Figura 5-7: Resultado tras borrar una unidad

Crear objetivo

Para crear un objetivo es necesario rellenar todos los campos obligatorios. Estos son: el número del objetivo, el título y el enunciado. Si alguno de estos campos está en blanco se mostrará un mensaje pidiendo que se rellene.

En esta prueba se creará un objetivo, con una pregunta y tres respuestas. También se añadirá un fichero de código.

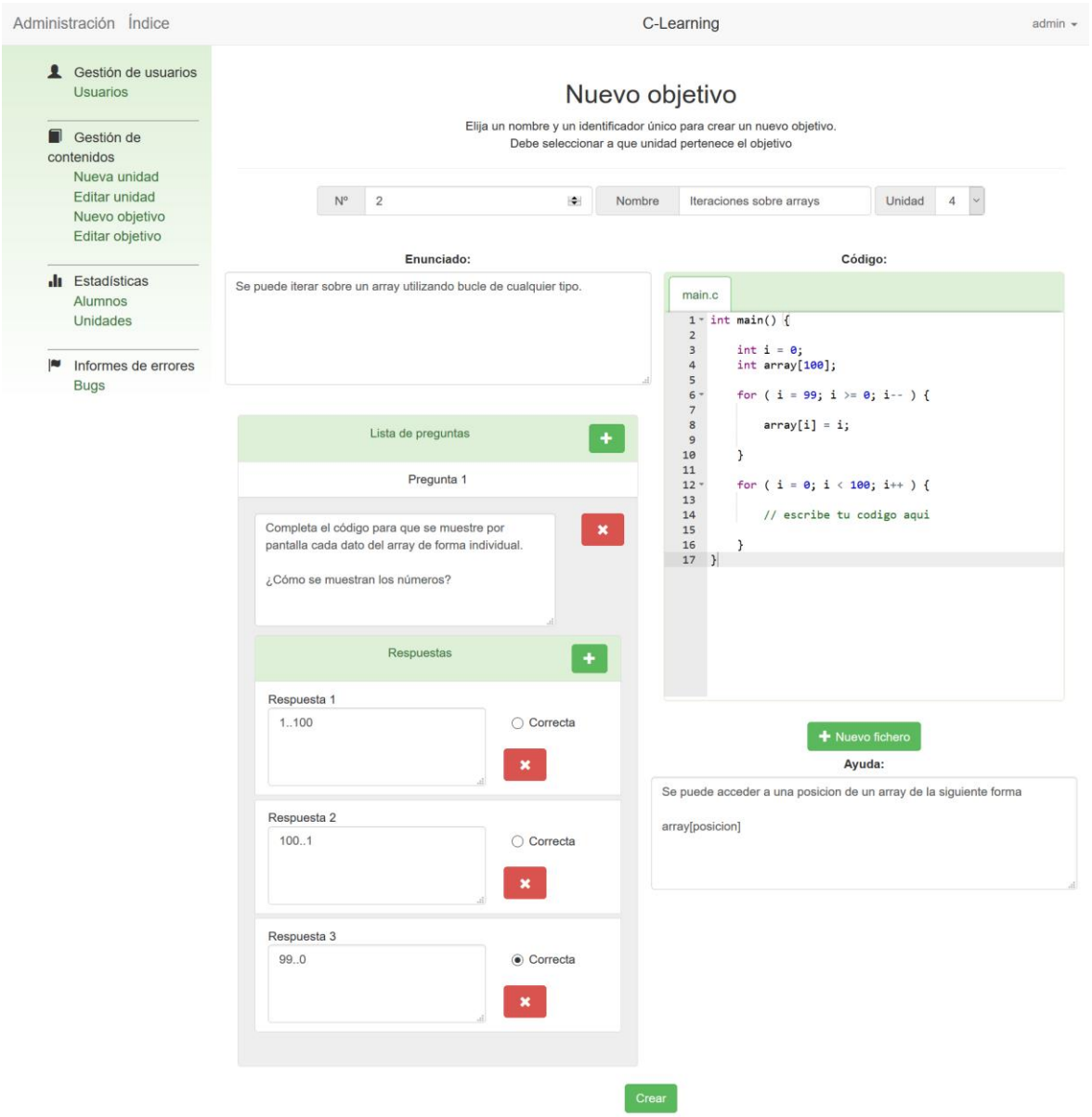


Figura 5-8: Creación de un objetivo

Para comprobar que se ha creado correctamente hay que seleccionar en el menú lateral editar objetivo. Aparecerá una lista con las unidades existentes. Como se ha marcado la unidad 4 para asignar el objetivo recién creado hay que seleccionar dicha unidad en la lista. A continuación, en el desplegable Ejercicio, hay que seleccionar el número de ejercicio que se ha asignado durante la creación, en este caso 2.

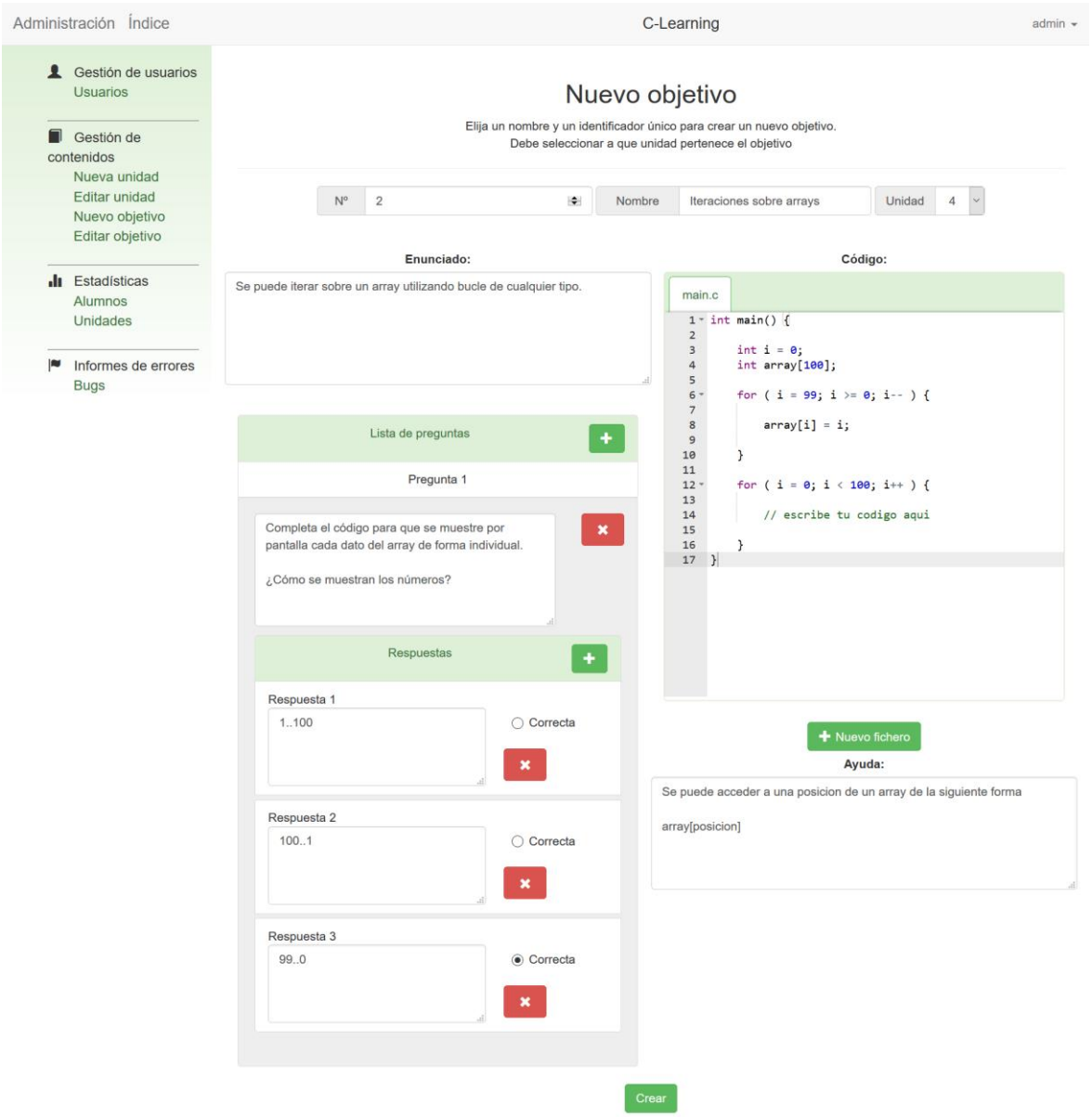


Figura 5-9: Comprobación de la creación de un objetivo

Como se puede comprobar el ejercicio se ha creado correctamente. La lista de preguntas aparece por defecto cerrada. Para desplegarla habrá que hacer clic sobre la pregunta que se desee consultar.

Edición de un objetivo

Existen multitud de posibilidades para editar un objetivo. En esta prueba se va a modificar el código y a añadir una pregunta con dos respuestas.

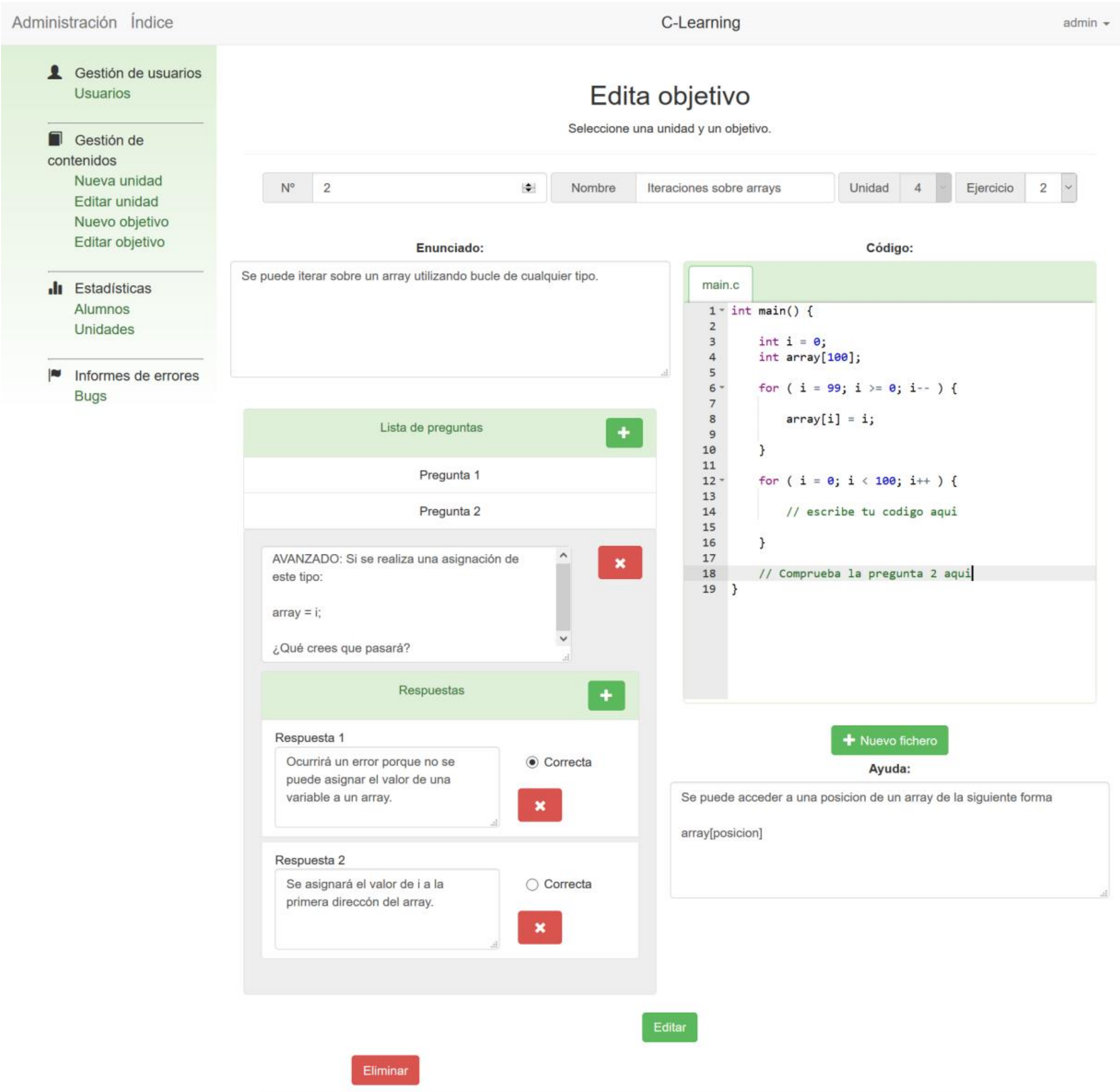


Figura 5-10: Modificación de un objetivo

Si se vuelve a cargar este objetivo a través del enlace editar objetivo se obtendrá el mismo resultado que se muestra en la Figura 5-10, así que hay que mirar en la base de datos para comprobar que efectivamente se han realizado las modificaciones.

```
mysql> mysql> select * from preguntas;
+----+-------------+--------------------------------------------------------------------------------+-----------+
| id | id_enunciado | texto                                                                         | nPregunta |
+----+-------------+--------------------------------------------------------------------------------+-----------+
|  1 |           1 | Compila el programa. Ejecútalo. ¿Qué aparece en la terminal?                   |         1 |
|  2 |           1 | Compila el programa. Ejecútalo. ¿Qué aparece en la terminal?                   |         2 |
|  3 |           1 | Compila el programa. Ejecútalo. ¿Qué aparece en la terminal?                   |         3 |
|  4 |           2 | Escribe la palabra hola con 4 variables idependientes.                         |         1 |
|  5 |           4 | ¿Qué tamaño en bytes tendrá el array del enunciado?                            |         1 |
|  7 |           6 | ¿Un argumento de entrada tiene el mismo nombre fuera de la función que dentro? |         1 |
| 15 |          15 | Busca en internet cuantas formas hay de hacer comentarios en C                 |         1 |
| 23 |          21 | Compila el programa. Ejecútalo. ¿Qué aparece en la terminal?                   |         1 |
| 28 |          26 | Completa el código para que se muestre por pantalla cada dato del array de forma individual.

¿Cómo se muestran los números?     |         1 |
| 29 |          26 | AVANZADO: Si se realiza una asignación de este tipo:

array = i;

¿Qué crees que pasará?                                          |         2 |
+----+-------------+--------------------------------------------------------------------------------+-----------+
10 rows in set (0.00 sec)
```

Figura 5-11: Comprobación de la modificación de preguntas

Estas son las pruebas básicas del módulo de edición.

5.2 Pruebas en el módulo de compilación

Para probar el correcto funcionamiento del módulo, hay que realizar distintas pruebas a cada uno de sus elementos.

- En primer lugar, es necesario compilar nuestro programa. Hay que probar los casos en los que el programa compila correctamente, aparecen warnings y cuando se encuentran errores.
- En segundo lugar, vamos a probar a introducir las sentencias especiales fopen y system para ver cómo se comportan.
- En tercer lugar, se va a probar a ejecutar un programa que no requiera de interacción por parte del usuario.
- En cuarto lugar, se probará un programa con interacción del usuario.
- En quinto lugar, se probará la funcionalidad de descargar.
- En sexto lugar, se probará a descargar el código que aparece en los editores.
- Por último, se probará a crear un nuevo fichero.

Compilación

Aquí se cubrirán los dos primeros puntos descritos anteriormente.

En primer lugar, se compilará un programa sencillo y sin errores ni warnings.

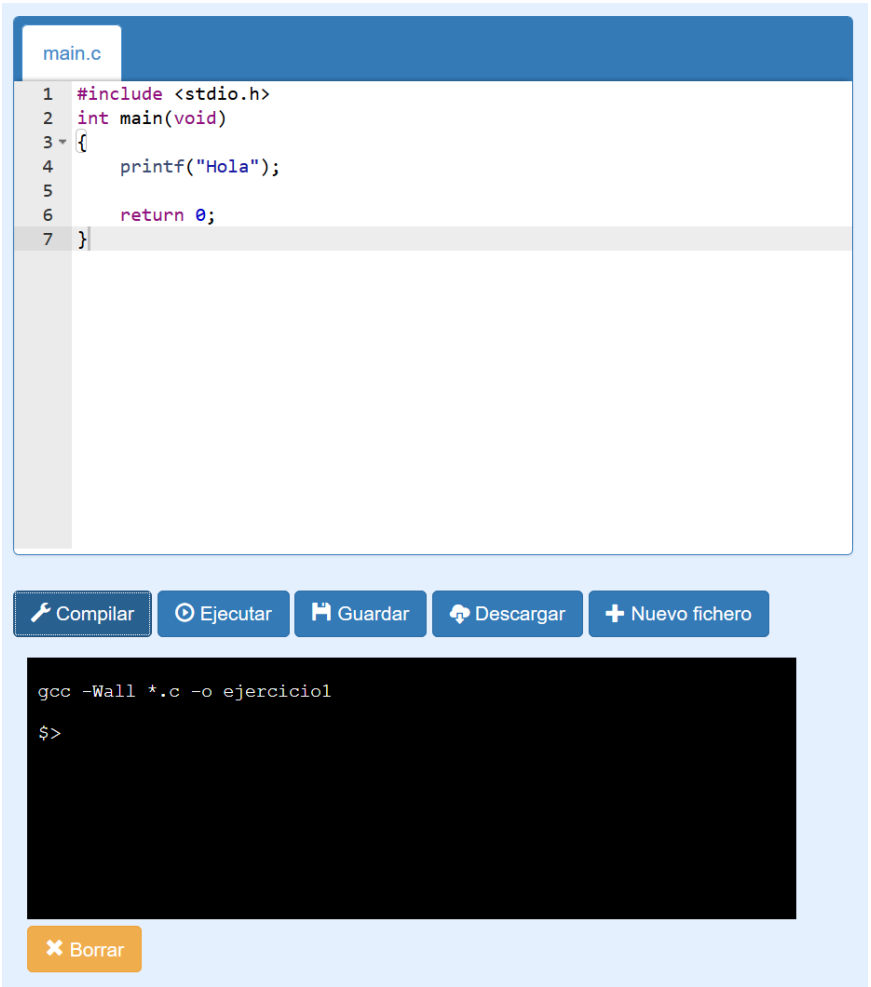


Figura 5-12: Compilación satisfactoria

Tras pulsar el botón compilar el resultado de la compilación se mostrará en el terminal.

Ahora se modificará el programa anterior para que produzca un warning. Se va a añadir una variable sin inicializar.

Figura 5-13: Compilación con warnings

Como se puede observar, efectivamente se muestra el *warning* y se señala correctamente donde se encuentra.

Para la siguiente prueba se va a eliminar el *warning* anterior y se añadirá un error en el código. En este caso se va a eliminar el punto y coma de la línea que contiene el *printf*.

Figura 5-14: Compilación con errores

Se puede comprobar que se muestra el error en la Figura 5-14.

Estas son las comprobaciones más básicas para ver si se puede compilar un programa. También es necesario comprobar las sentencias especiales que pueden aparecer en el código y que se han descrito en la sección 4.2.1. En primer lugar, se va a probar el funcionamiento de *system*.

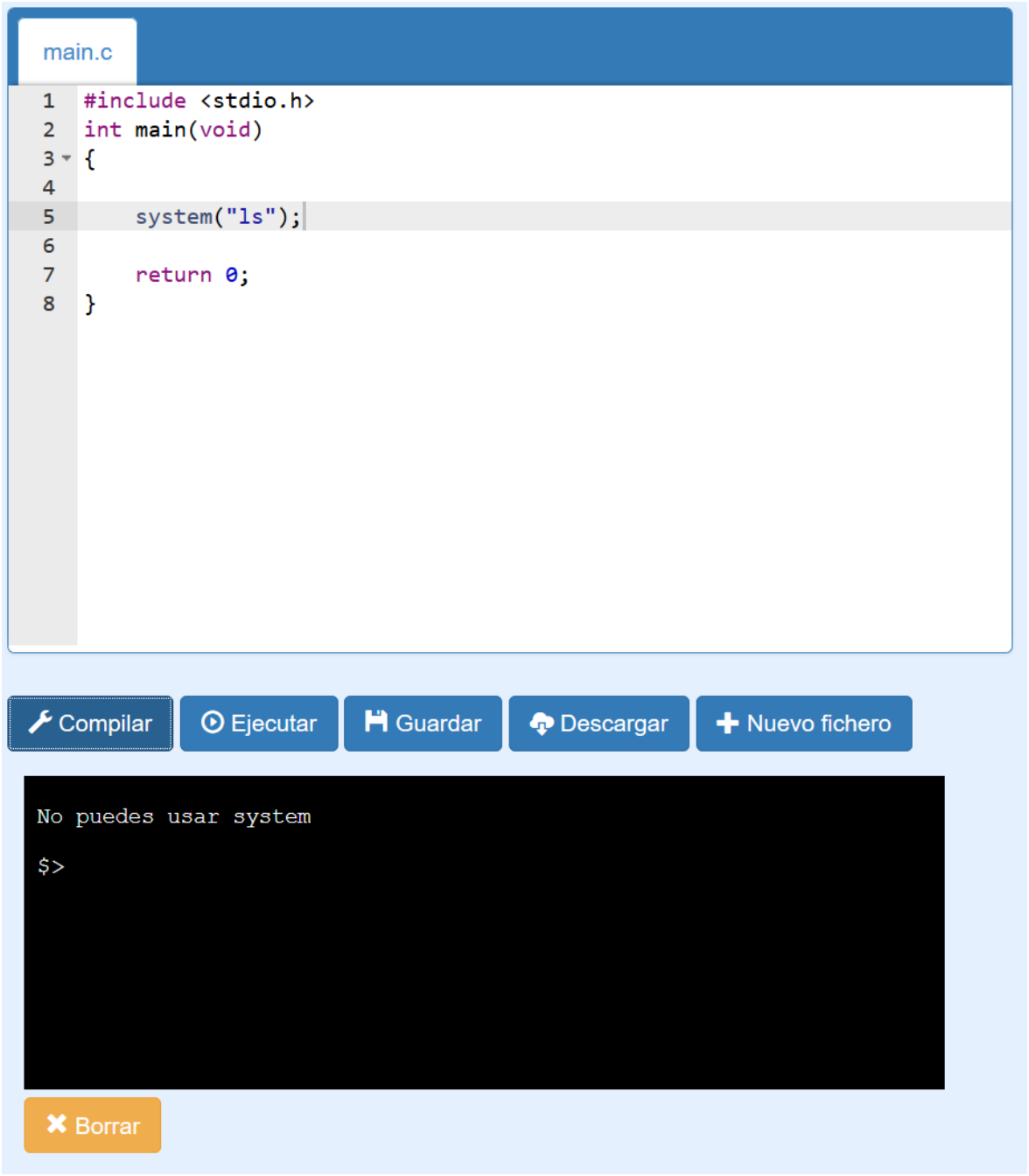


Figura 5-15: Compilación con una instrucción system

Como se puede observar en el terminal, la compilación produce un mensaje diciendo que no se puede usar *system*. Esto es como se ha descrito en secciones anteriores por motivos de seguridad.

Ejecución

Para la primera prueba de ejecución, se escoge un programa sencillo, que simplemente muestre un mensaje por pantalla con el resultado mostrado en la figura 5-16:

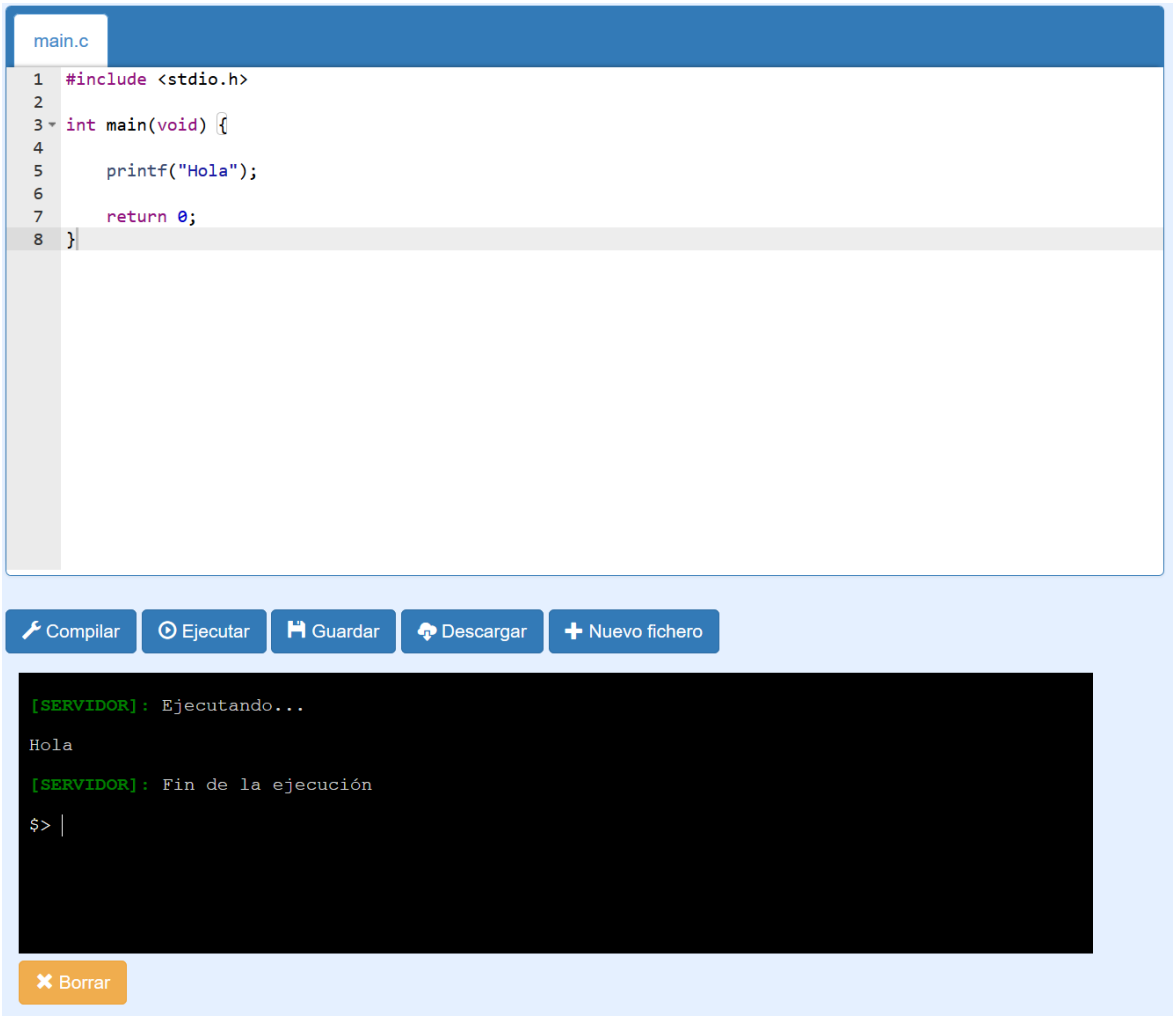


Figura 5-16: Ejecución sin interacción por parte del usuario

Como se puede observar, el servidor informa al usuario cuando comienza una ejecución y cuando termina.

En la siguiente prueba se va a probar una ejecución interactiva. Se le va a requerir al usuario que introduzca un número mayor que cero.

Figura 5-17: Ejecución. El programa espera la interacción del usuario

Como se puede observar en la figura 5-17, el programa espera a que el usuario introduzca un número. Si una ejecución se encuentra en espera es necesario bloquear los botones de ejecutar y compilar. De esta forma se evitan posibles errores que puedan surgir de modificar el ejecutable que, a su vez, se está ejecutando.

Se introduce un cero:

Figura 5-18: Ejecución. El usuario introduce un número.

Como cero no es mayor que cero el programa vuelve a pedir que se introduzca un número. Se introduce un tres:

Figura 5-19: Ejecución. El usuario introduce otro número

Como tres es mayor que cero el programa sale del bucle y termina su ejecución.

Por último, se va a hacer una prueba más de ejecución. Esta prueba corresponde al correcto funcionamiento de *fopen*. Se va a generar un fichero en tiempo de ejecución.

Figura 5-20: Ejecución con fopen

Para comprobar que se ha creado basta con mirar directorio del usuario y su carpeta correspondiente (Figura 5-21):

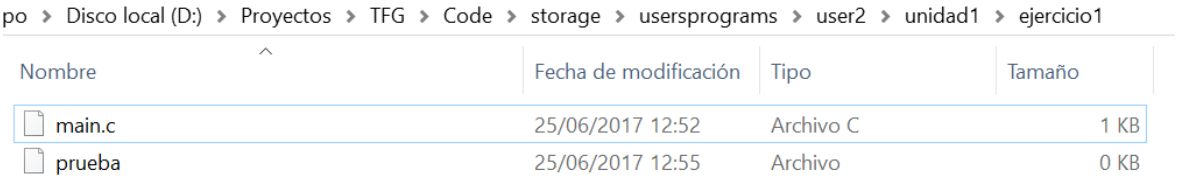


Figura 5-21: Ejecución. Se comprueba que se ha creado el fichero

Guardar, descargar y crear nuevo fichero

Para comprobar el funcionamiento de **guardar** se va a modificar un fichero, se va a pulsar el botón guardar y se va a comprobar el fichero mismo. Se escoge como base el fichero que se muestra en la figura 5-22:

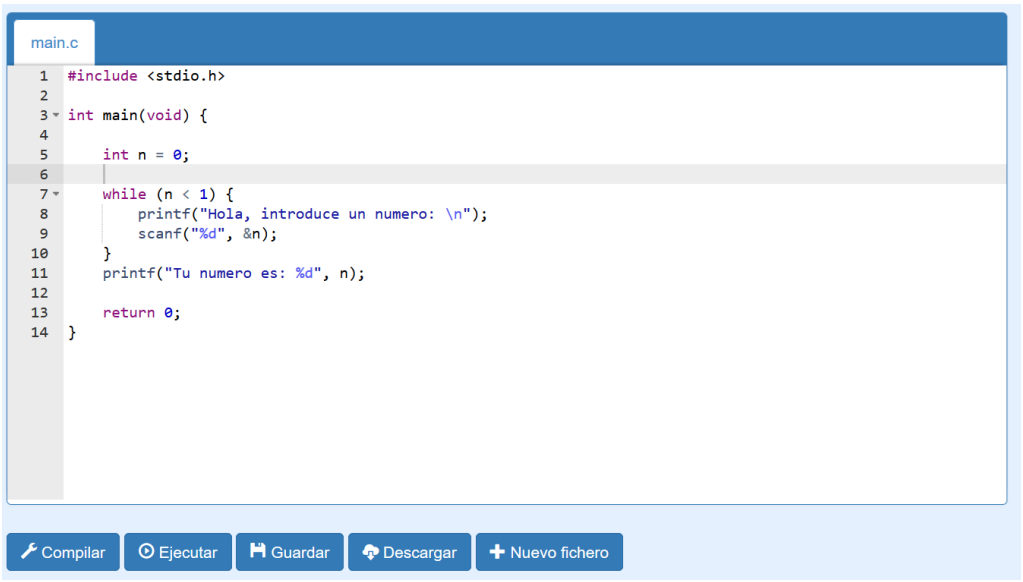


Figura 5-22: Guardar. Estado inicial del fichero

Se modifica y se guarda:

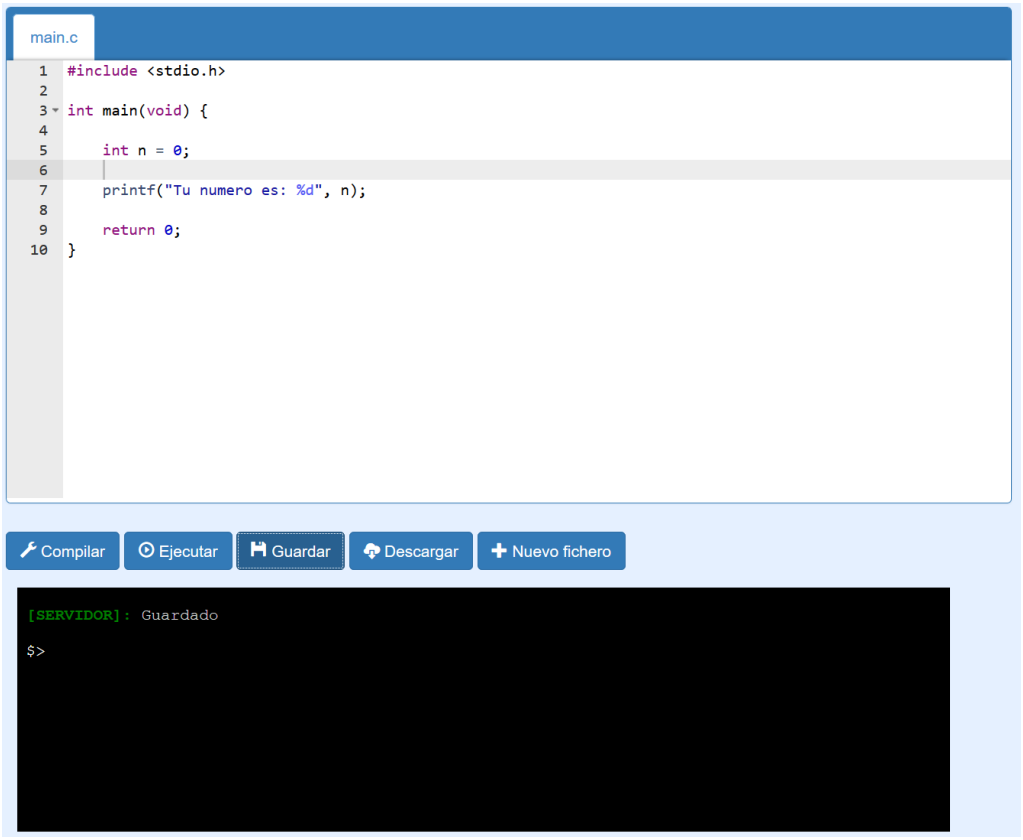


Figura 5-23: Guardar. Se modifica el fichero y se guarda

El servidor informará en el terminal que se ha guardado. Lo comprobamos en la carpeta del usuario:

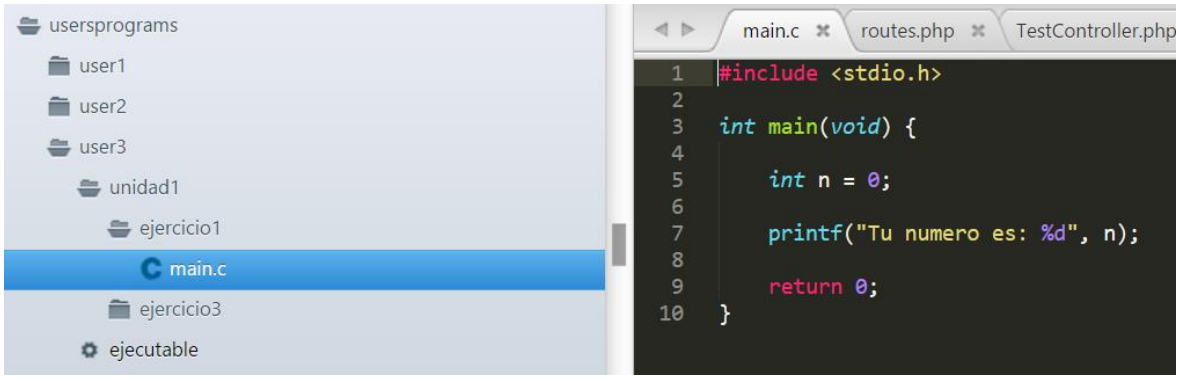


Figura 5-24: Guardar. Se comprueba que el fichero se ha guardado

Para probar la generación de ficheros mediante la interfaz hay que introducir un nombre para el fichero, y una extensión. Se va a generar un fichero llamado library.h.

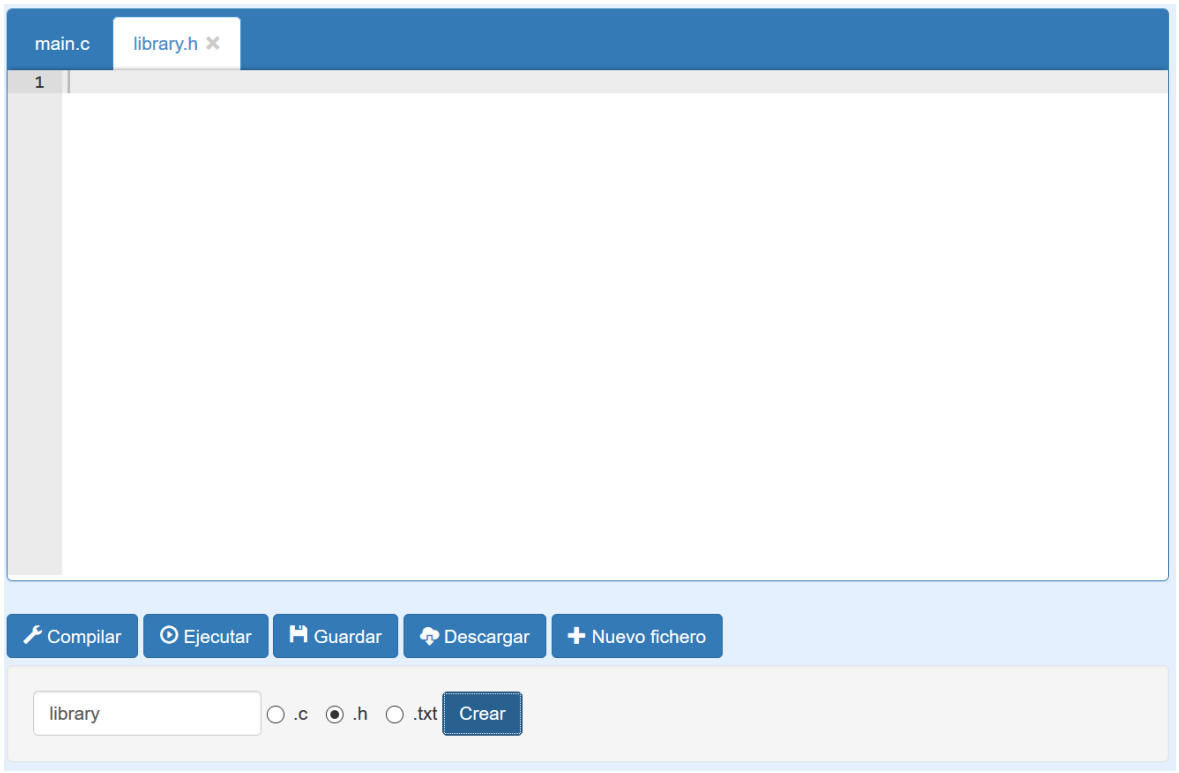


Figura 5-25: Creación de un nuevo fichero

Hay que tener en cuenta que hasta que no se pulse otra vez guardar, el fichero no se creará en el servidor.

Por último, se va a probar a **descargar**. En este caso se descargarán los ficheros main.c y library.h (se ha guardado de la prueba anterior):

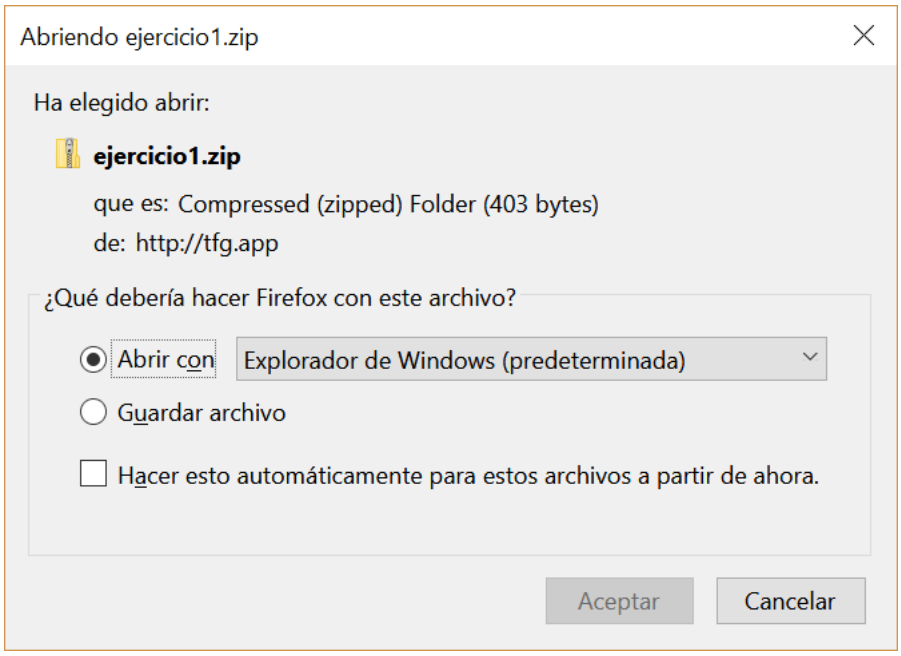


Figura 5-26: Descarga

Se puede comprobar que se descarga un fichero .zip con el número del ejercicio. Comprobamos el interior del zip para ver que efectivamente contiene los dos ficheros.

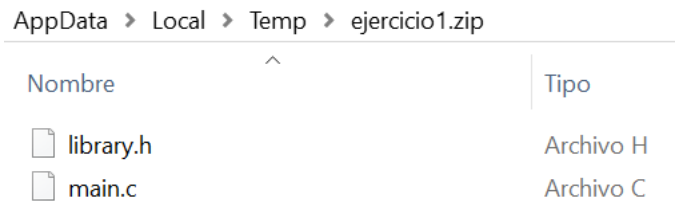


Figura 5-27: Comprobación de los ficheros descargados

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Tras la realización de este trabajo se pueden sacar varias conclusiones basándonos en lo objetivos cumplidos.

El trabajo se ha realizado a partir de un proyecto en desarrollo, en el que han trabajado varias personas anteriormente. Se ha podido estudiar su diseño para adaptarlo a los nuevos requisitos realizando los cambios necesarios en los módulos ya desarrollados para, posteriormente, añadir la nueva funcionalidad. Esto es fundamental, porque en muchos casos en el mundo profesional es probable encontrarse en una situación similar.

Se ha conseguido mejorar el diseño para hacerlo más escalable y modular. De esta forma si en un futuro se decide modificar el trabajo se podrán realizar cambios de forma sencilla y sin que afecte a toda la estructura del proyecto. También se ha añadido documentación en el código de forma que ayude en esos posibles cambios.

Se ha realizado un nuevo módulo de compilación y ejecución que permitirá realizar dichas tareas a nuevos estudiantes en su primer contacto con la programación.

Por último, me gustaría destacar el aprendizaje sobre el desarrollo utilizando un framework, en este caso Laravel, que a pesar de que en un principio requiere de un gran esfuerzo para comprender el funcionamiento del mismo, una vez que se domina permite realizar el trabajo de forma sencilla y eficaz.

6.2 Trabajo futuro

En cuanto al trabajo futuro, existen multitud de posibilidades para ampliar la aplicación, estas son algunas de ellas:

- Es posible mejorar la gestión de los usuarios. Actualmente solo se guardan los datos básicos del usuario para que funcione la aplicación. Sería interesante añadir otros datos como el *nia*, utilizado en las plataformas de la universidad, o los apellidos. Más interesante aún sería conectar la aplicación con los servicios de la universidad de forma que la creación de la cuenta de un usuario se pueda realizarse al darse de alta en el curso correspondiente de Moodle.
- Por otro lado, el objetivo global de la aplicación es reducir la curva de aprendizaje en el primer curso para el primer lenguaje de programación que se estudie. Sería muy práctico mejorar el sistema de estadísticas que incluye la aplicación, generando nuevos datos como por ejemplo el número de errores se que comenten al responder las preguntas organizados por ejercicios. Esto permitiría discernir la dificultad que puede tener un ejercicio y si se está explicando de forma adecuada.
- No solo es importante generar estadísticas de forma automática, la opinión de los alumnos es fundamental. De igual forma que en Moodle existe un sistema de encuestas, sería interesante añadir alguna forma de valorar cada objetivo.
- Otra mejora podría consistir en la creación de posibles controles o exámenes al final de cada unidad.

- Por último, es posible añadir algunos sistemas de seguridad adicionales. Por ejemplo, forzar a la aplicación a utilizar siempre HTTPS en lugar de HTTP en sus peticiones al servidor. Otro aspecto de seguridad que se puede añadir es en base a la ejecución de los programas de los usuarios. Se pueden utilizar contenedores para controlar dicha ejecución y realizarla en un entorno completamente seguro.

Referencias

[1] Richard M. Stallman and the GCC Developer Community, “Using the GNU Compiler Collection (GCC)”, GNU Press. [En línea] 20/06/2017. https://gcc.gnu.org/onlinedocs/gcc-7.1.0/gcc/Standards.html#C-Language

[2] “Prácticas de sistemas operativos”. [En línea] 20/06/2017. http://1984.lsi.us.es/wiki-practicas-ssoo/index.php/Gcc

[3] “Cygwin Oficial Web Page”. [En línea] 23/06/2017. https://www.cygwin.com

[4] “Relationship to the World Wide Web and REST Architectures”, W3C Web Services Architectures. [En línea] 22/06/2017. https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest

[5] “Bootstrap 3 Tutorial”, w3schools. [En línea] 20/06/2017. https://www.w3schools.com/bootstrap/default.asp

[6] “Laravel Homestead” [En línea] 22/06/2017. https://laravel.com/docs/5.2/homestead

[7] “webscoketd”, GitHub, Joe Walnes. [En línea] 22/06/2017. https://github.com/joewalnes/websocketd

[8] A. Melnikov, “The WebSocketProtocol”, IETF, December 2011. [En línea] 22/06/2017. https://tools.ietf.org/html/rfc6455#page-5

[9] “Ace Official Web Page”. [En línea] 26/07/2017. https://ace.c9.io/#nav=about&api=document

[10] “WebSockets”, Mozilla Developer Network. [En línea] 22/07/2017. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[11] “Laravel Database: Migration”. [En línea] 20/07/2017. https://laravel.com/docs/5.2/migrations

[12] “Laravel Database: Query Builder”. [En línea] 20/07/2017. https://laravel.com/docs/5.2/queries

Glosario

API	Application Programming Interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
IDE	Integrated Development Enviroment
MVC	Modelo Vista Controlador
PHP	PHP: Hypertext Preprocessor
REST	Representational State Transfer
SQL	Structured Query Language
URI	Uniform Resource Identifier

Anexos

A Manual del programador

Configuración del entorno de desarrollo
Laravel ofrece un entorno de desarrollo completo y listo para casi empezar a desarrollar. Homestead es una máquina virtual con todo el software básico y necesario. Hay que mencionar que la versión de Laravel utilizada en el proyecto es la 5.2. Para instalar la máquina basta con consultar la documentación de Laravel [6].

Para configurar el entrono existe un fichero *Homestead.yaml* que es necesario modificar.
En primer lugar, hay que definir los parámetros de la máquina virtual. En primer lugar, la ip de la máquina, el número de CPUs y el tamaño de la memoria principal. Es recomendable que estos valores se asemejen al entorno de producción, pero si no es posible pueden ser menores.

A continuación, hay que vincular las carpetas de la máquina virtual con las del equipo host. Un ejemplo es el que aparece en la Ilustración 1.

```
ip: "192.168.10.10"
memory: 4096
cpus: 2
provider: virtualbox

authorize: ~/.ssh/id_rsa.pub

keys:
    - ~/.ssh/id_rsa

folders:
    - map: d:/Proyectos/TFG/Code
      to: /home/vagrant/Code

sites:
    - map: tfg.app
      to: /home/vagrant/Code/public

databases:
    - desarrollo
```

Ilustración 1: Homestead.yaml

La carpeta */home/vagrant/Code* de la máquina virtual se sincronizará con la carpeta del equipo host *d:/Proyectos/TFG/Code*.

También es necesario añadir el nombre de la aplicación. Este nombre se utilizará para acceder desde el navegador, y es necesario añadirle el punto de entrada que siempre será la carpeta *public*.

Por último, hay que definir el nombre de la base de datos que en este caso es *desarrollo*.

Estructura de directorios

A la hora de desarrollar un proyecto en Laravel hay que tener en cuenta donde se encuentran los distintos recursos del proyecto.

- *App*: En esta carpeta se encuentra todo el código que se ejecuta en el lado del servidor. Dentro, existen bastantes carpetas, pero las más importantes son:
    - *Classes*: Aquí se encuentran las clases definidas.
    - *Http*: Aquí se encuentran los controladores de las vistas. Donde se controlan las peticiones que llegan al servidor. En esta carpeta también se encuentra el fichero *routes.php* del que se hablará un poco más abajo.
- *Config*: Esta carpeta contiene toda la configuración del proyecto. En un principio no es necesario tocar nada aquí, pero es interesante ver que contiene.
- *Database*: Esta carpeta contiene las operaciones relacionadas con gestión de la base de datos. Si se necesita modificar la base de datos es interesante hacerlo a partir de las migraciones.
- *Public*: Esta es la carpeta de entrada. Esto quiere decir que todo se ejecuta desde aquí. Hay que tenerla como referencia si se quieren abrir ficheros o crearlos. Esta carpeta contiene también los ficheros CSS y JavaScript del proyecto, así como las imágenes.
- *Resources*: En esta carpeta se encuentra otra muy importante, *views*. Dentro de *views* se encuentran todos los ficheros con código HTML, esto es, las vistas. También contiene sus plantillas.
- *Storage*: Esta carpeta contiene la carpeta *userprograms* que es donde se guaran todos los ficheros de los usuarios.

Vistas

Las vistas son ficheros que contienen la estructura HTML. Estos ficheros se consideran recursos y se cargarán mediante URIs. A una vista se le pueden pasar argumentos de entrada de la siguiente forma:

```
view('test')->with($data);
```

donde data puede ser un array y test es el nombre de la vista.

Si se quiere que varias páginas contengan elementos comunes, un menú lateral, por ejemplo, se pueden usar plantillas. Las plantillas se encuentran dentro de la carpeta *layauts*.

Controladores

Los controladores contienen la funcionalidad para responder a peticiones de los clientes. La llamada los controladores se hace desde el fichero routes.php. Este fichero es muy importante porque asigna una ruta a un recurso. Por ejemplo:

```
Route::post('/editaUnidad/borrar', 'GestorUnidadController@borraUnidad');
```

En esta línea de código se observan tres elementos diferenciados:

- Post: es el tipo de petición que se hace al servidor.
- /editaUnidad/borrar: es la URI o ruta.
- GestorUnidadController@borraUnidad: hace referencia al método borraUnidad que está contenido en el controlador GestorUnidadController.php

Para acceder a los parámetros enviados a un controlador se utiliza la variable request. Veamos un ejemplo:

```
public function borraUnidad(Request $request) {

    $nUnidad = $request->nUnidad;

    //Resto de código de la función

}
```

Como podemos ver en el ejemplo $request contiene un parámetro llamado nUnidad. Igualmente puede contener cualquier número de parámetros. Veamos la petición del cliente a esta función:

```
<form method="post" action="/editaUnidad/borrar" class="form-horizontal">
    <div class="col-xs-4">
        <input type="hidden" name="_token" value="{{ csrf_token() }}">
        <input id="nUnidadBorrar" type="number" class="form-control"
            name="nUnidad"
            placeholder="Introduza un número"
            min="1" max="100" style=" display: none;">
        <button type="submit" class="btn btn-danger"> Eliminar
        </button>
    </div>
</form>
```

Analizemos los elementos resaltados. Post es el tipo de petición, definida en el fichero routes.php, /edita/Unidad/borrar es la URI del recurso, también definida en el fichero routes. _token y nUnidad son los parámetros que se envían al controlador, y en este caso al método borraUnidad.

Nota: el parámetro _token debe estar en todas las peticiones ya que identifica al usuario registrado. Si en algún momento apareciese algún error en el que se haga referencia al token y este esté incluido en la petición habrá que borrar las cookies del navegador asociadas a la aplicación.

Documentación del código

Es importante que el código sea claro y se pueda leer fácilmente, así como que los métodos vengan acompañados de una cabecera para saber que hacen, que tipo de parámetros necesitan y que tipo de retorno devuelven. Esto es importante porque tanto PHP como JavaScript no define variables con tipos. Un ejemplo de cabecera de un método:

```
/**----------------------------------------------------------------------------
 *
 * Función: compilar
 *
 * Descripción:
 *      - Guarda el contenido de los editores en los ficheros.
 *      - Analiza el código.
 *      - Compila si no hay problemas.
 *      - Devuelve el resultado de la compilación.
 *
 * Parámetros:
 *      Request $request : contiene la petición POST
 *
 * Retorno:
 *      Devuelve la salida de texto tras la compilación o un mensaje de error.
 *
 *----------------------------------------------------------------------------*/
```

B Manual de instalación en Ubuntu

Requisitos

- LAMP
- Git
- Composer
- Laravel (Proyecto)

Instalar LAMP en Ubuntu

El paquete LAMP es un conjunto de herramientas para Linux que incluye Apache como servidor web, MySQL como motor de bases de datos relacionales y PHP. Para instalar LAMP en Ubuntu será necesario abrir un terminal y ejecutar los siguientes pasos:

```
$ sudo apt-get update
$ sudo apt-get install lamp-server^
```

Durante la instalación se pedirá definir una contraseña para el usuario root de MySQL.

Configuración de Apache

Es una buena práctica añadir un directorio en la carpeta home que esté directamente vinculada con nuestra aplicación web. Para ello:

- Se creará una carpeta llamada *public_html* en el directorio home.
- Se vinculará con la carpeta del servidor de la siguiente forma:

```
$ sudo ln -s /home/$USER/public_html /var/www
```

En segundo lugar, hay que decirle a nuestro servidor donde estará la ruta de entrada a la aplicación, esto es el fichero *index.php*. Siguiendo la estructura de directorios de Laravel, este se encuentra en la carpeta *public* por lo tanto será necesario indicar a nuestro servidor que se encuentra ahí de la siguiente forma:

```
$ sudo nano /etc/apache2/sites-available/my_app.conf
```

> **Nota**: El fichero my*_app.conf* no existe de inicio. Se puede crear o utilizar el fichero que viene por defecto. En nuestro caso 000-default.conf.
> Si no se utiliza el fichero por defecto se deberá habilitar el nuevo de la siguiente forma:
>
> ```
> $ sudo a2ensite my_app
> ```

Dentro de este fichero será necesario modificar lo siguiente:

DocumentRoot /var/www/your-project-name/public

<Directory /var/www/your-project-name/public>
Options -Indexes +FollowSymLinks +MultiViews

AllowOverride All

Require all granted
</Directory>

Por último, será necesario reiniciar Apache para cargar la nueva configuración.

```
$ sudo service apache2 reload
```

Instalación de Git y Composer
Para instalar Git basta con ejecutar lo siguiente:

```
$ apt-get install git-all
```

Para instalar Composer será necesario seguir los siguientes pasos:

```
$ php -r "copy('https://getcomposer.org/installer', 'composer-
setup.php');"

$ php composer-setup.php

$ php -r "unlink('composer-setup.php');"

$ sudo apt install composer
```

Instalar nuestra aplicación Laravel
Para instalar nuestra aplicación sólo será necesario copiarla entera en la carpeta que hemos creado anteriormente, *public_html* y dar los permisos necesarios a la carpeta *storage*:

```
$ chmod 777 -R storage
```

Configuración de la base de datos MySQL
Para configurar la base de datos es necesario seguir los siguientes pasos:

- Crear la base de datos.
- Importar las tablas desde el fichero *db_schema.sql*.
- Configurar la aplicación Laravel con nuestra base de datos.

Crear la base de datos
Se accederá a MySQL de la siguiente forma:

```
$ mysql -u <usuario> -p
```

Es necesario introducir la contraseña para el usuario que creamos anteriormente (root por defecto).
Una vez dentro creamos la base de datos:

```
mysql> CREATE DATABASE <nombre_db>
```

Importar las tablas
Solo será necesario introducir el siguiente comando:

```
$ mysql -u <usuario> -p < db_schema.sql
```

E introducimos la contraseña del usuario.

> **Nota**: la primera línea del fichero *db_schema.sql* contiene el nombre de la base de datos. Este deberá actualizarse con el que le hayamos dado anteriormente.

Configurar la base de datos con nuestra aplicación
La configuración consiste en modificar el *fichero .env* que se encuentra en el directorio raíz (está oculto) de la aplicación. Será necesario modificar las siguientes variables:

```
DB_DATABASE=<nombre_db>
DB_USERNAME=<usuario>
DB_PASSWORD=<password>
```